Commodore
Sachbuch

Frank Riemenschneider

AMIGA
Programmieren in Maschinensprache

Ein modularer Programmierkurs mit dem Devpac-Assembler

68000-Prozessor ★ Intuition ★ Fließkommarechnung
★ Grafikprogrammierung ★ Ein-/Ausgabe mit Amiga-DOS ★ Devices

Alle im Buch vorgestellten Beispielprogramme
sind auf Diskette enthalten.

AMIGA
Programmieren in Maschinensprache

AMIGA

Programmieren in Maschinensprache

Ein modularer Programmierkurs mit dem Devpac-Assembler

68000-Prozessor ★ Intuition ★ Fließkommarechnung ★ Grafikprogrammierung ★ Ein-/Ausgabe mit Amiga DOS ★ Devices

Frank Riemenschneider

Commodore Sachbuch

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Riemenschneider, Frank:
Amiga - Programmieren in Maschinensprache : ein modularer Programmierkurs mit dem Devpac-Assembler /
Frank Riemenschneider. - Haar bei München : Markt-u.-Technik-Verl., 1989
(Commodore-Sachbuch)
ISBN 3-89090-712-1

15 14 13 12 11 10 9 8 7 6 5 4 3 2
92 91 90 89

ISBN 3-89090-712-1

Einbandgestaltung: Grafikdesign Heinz Rauner
Dieses Buch wurde mit Desktop Publishing bearbeitet
und auf Linotronic 300 belichtet.
Druck: Kösel, Kempten
Printed in Germany

Inhaltsverzeichnis

Maschinensprache auf dem Amiga - vorsintflutlich oder fortschrittlich?

Vorwort

Wenn Sie wie ich, liebe Leser, einmal in einer so schnellen und komfortablen Sprache wie Turbo Pascal auf einem AT programmiert haben, werden Sie wahrscheinlich nicht daran denken, jemals noch einmal auf Maschinensprache zurückzukommen, zumal auf einem Superrechner wie dem Amiga!

Genau dieses hatte ich mir nach dem Umstieg vom C64 auch vorgenommen. Das Problem besteht jedoch darin, daß auf dem Amiga keine vernünftigen Alternativen zur Maschinensprache existeren. Sie glauben das nicht? Nun, sehen wir sie uns doch einmal an:

Als erstes wäre das Amiga-Basic zu nennen, eine Interpretersprache. Sie weist gegenüber der Maschinensprache gravierende Nachteile auf: Zum einen ist ein Basic-Programm auf Grund der Übersetzung bis zu 100mal (!) langsamer als ein entsprechendes Maschinenprogramm. Dieses grundsätzliche Problem wird beim Amiga-Basic noch dadurch verschärft, daß der Interpreter nicht in Maschinensprache sondern in »C« geschrieben ist, die maximale durch die Hardware bestimmte Geschwindigkeit also niemals erreichen kann. Des weiteren benötigt der Interpreter selbst eine Menge Speicherplatz (103 500 Byte), die dem Anwender fehlen kann, von der lästigen Ladezeit von der Diskette ganz zu schweigen. Diese Probleme sind aber einem gegenüber fast vernachlässigbar: Es ist nicht möglich, mit den Basic-Befehlen alle Möglichkeiten des Amiga auszunutzen. Besonders die Programmierung der grafischen Benutzeroberfläche (Intuition) ist durch das Fehlen der Gadgets und Untermenüs schwierig bis unmöglich.

Zusammenfassend kann man feststellen, daß das Amiga-Basic denkbar ungeeignet ist, einen Rechner wie den Amiga zu programmieren. Sehen wir uns deshalb gleich die nächste Hochsprache, »C«, an. Im Gegensatz zum Basic handelt es sich hier um eine Compilersprache, d.h., das Programm wird vor dem Ablauf in ein Maschinenprogramm übersetzt. Nun, von keinem Kompilat kann man erwarten, daß es genauso kurz und schnell ist wie ein reines Maschinenprogramm, weil der Compiler natürlich immer universelle Routinen integrieren muß. Auf dem Amiga ist für mich aber das Zumutbare überschritten: Vor allen Dingen dadurch, daß das Betriebssystem in »C« geschrieben wurde, treten des öfteren Zeitprobleme auf. Zum anderen ist die Sprache gegenüber Basic relativ maschinenorientiert, d.h. man kann nicht mehr sicher sein, daß alle Programmierfehler abgefangen werden können. Vielmehr muß man wie bei der Maschinensprache mit Systemabstürzen rechnen. Das Problem dabei ist, daß ein Absturz immer auf Grund eines Maschinenbefehls erfolgt.

Deshalb muß man das Kompilat analysieren, um den Grund für den Absturz zu entdecken. Diese Arbeit kann man seinem ärgsten Feind nicht wünschen. Helfen können sogenannte symbolische Debugger, bei denen zu jedem Maschinenbefehl der entsprechende »C«-Befehl angezeigt wird, ohne Maschinensprachekenntnisse kommt man aber auch dort nicht weiter. Schon die Programmentwicklung kann jedoch relativ nervtötend sein, da die Kompilierung relativ lange dauert und bei jedem gefundenen Fehler wiederholt werden muß. Schließlich muß man auch die hohen Kosten für den Compiler und den Debugger nennen.

Damit sind wir auch schon bei der Maschinensprache. Sie ist die einzige Sprache, die direkt vom Prozessor verstanden wird und daher nicht übersetzt zu werden braucht. Deshalb bietet nur sie maximale Geschwindigkeit sowie kompakte Programme, wie man sie auch mit dem besten Compiler nicht erreichen kann. Die Geschwindigkeit ist beim Amiga besonders bedeutend, da das Betriebssystem selbst ein C-Kompilat und daher nicht so schnell ist. Mit dem »DEVPAC« steht ein Entwicklungssystem zur Verfügung, das die Erstellung von Maschinenprogrammen mindestens so komfortabel macht wie ein »C«-Programm, wobei die Assemblierungszeiten vernachlässigbar klein sind. Durch den Einsatz des Debuggers »Monam« wird jede Fehlersuche zum Kinderspiel, zumal alle Guru-Meditationen abgefangen werden. Für den DEVPAC spricht auch der günstige Preis (148.- DM), in dem sowohl Assembler als auch Debugger enthalten sind.

Nach einer Umfrage des Amiga-Magazins im Juni 1988 programmierten zu diesem Zeitpunkt 65% der Leser in Basic, 14% in Maschinensprache und 7% in »C«. Dabei wünschen sich fast alle Programmierer, in Maschinensprache programmieren zu können. Diese Diskrepanz kann man nicht damit erklären, daß die Maschinensprache angeblich schwer zu erlernen sein soll – sie ist nur »anders« aufgebaut. Der Grund liegt meiner Meinung nach in dem Aufbau vieler Assemblerbücher: Im Gegensatz zu z.B. Basic- oder »C«-Büchern scheuen sich viele Autoren, relativ komplexe Programme darin zu veröffentlichen, sondern drucken viele kleine Beispiele ab, die in keinerlei Zusammenhang stehen. Ich bin jedoch der Ansicht, daß man als Leser mehr davon hat, ein Listing zu studieren, als endlose Seiten Theorie zu lesen, deren Anwendung im Dunklen bleibt. Bestätigt wird diese Meinung von vielen Amiga-Besitzern, die sagen, daß es ihnen mehr bringt, ein zusammenhängendes Programm aus dem Amiga-Magazin zu analysieren, als ein kleines Beispiel-Programm aus einem Buch!

Wie schon in meinem Buch »C64/C128: Alles über Maschinensprache« bin ich daher dazu übergegangen, von den üblichen Mini-Programmen, die immer nur einen Aspekt des Rechners erläutern können, auf »richtige« Programme überzugehen, die auch das Zusammenspiel von z.B. Windows, Grafiken, Menüs, etc. aufzeigen. Besonderer Wert wurde darauf gelegt, daß die dort verwendeten Routinen wirklich universell einsetzbar sind und damit leicht von Ihnen in Ihre eigenen Programme übernommen werden können.

Nach einer umfangreichen Einführung in die Maschinensprache des 68000 werden deshalb gleich die wichtigsten Themen in Angriff genommen, wobei reger Gebrauch von den Library-Funktionen gemacht wird: Im Kapitel über Intuition wird gezeigt, wie man durch die Verwendung von Menüs, Gadgets, etc. ein bedienerfreundliches, mausgesteuertes Pro-

gramm entwerfen kann. Besonders ausführlich wird dann – dem Amiga angemessen – die Grafik behandelt. Vom Setzen eines Punktes über die Verwendung von Fremdzeichensätzen bis zur superschnellen Ellipsenroutine durch direkten Zugriff auf die Grafik-Bitplanes wird alles gezeigt. Ein Beispielprogramm zur Erstellung von Chart-Grafiken in Form von 3-D-Torten- und Säulengrafiken mit Beschriftungen durch beliebige Zeichensätze schließt diesen Abschnitt ab.

Im folgenden Kapitel wird (zum erstenmal überhaupt in einem Buch) auf das Rechnen mit Fließkommazahlen eingegangen. Vom Aufbau dieser Zahlen, dem Rechnen mit den Mathematik-Librarys sowie der Ein- und Ausgabe wird dann die Rede sein. Am Ende steht eine total mausgesteuerte Luxus-Wertetabelle als Beispielprogramm.

Auch der Ein-/Ausgabe mit dem Amiga-DOS wird ein Kapitel gewidmet. Dort geht es hauptsächlich um die Verwaltung von Diskettendateien und den Lese-Schreib-Zugriff darauf. Ein weiteres Beispielprogramm demonstriert, wie professionelle Programme wie z.B. DeluxePaint beim Laden und Speichern eine Fileauswahl per Maus realisieren. Dort wird auch gezeigt, wie man IFF-Grafiken auf der Diskette ablegen kann oder diese von anderen Grafikprogrammen übernehmen kann.

Schließlich folgt ein Kapitel über Devices. Hiermit kann man nicht nur direkt z.B. auf jeden Diskettensektor einzeln zugreifen, sondern auch Bildschirmhardcopies in allen Größen und Formen ausdrucken. Das Beispielprogramm dieses Abschnittes erzeugt daher einen überdimensionalen Ausdruck. Dieses Programm können Sie aber auch dazu verwenden, einen Bildschirm im Briefmarkenformat auszugeben.

Im Anhang schließlich finden Sie neben allen Libraries, Strukturen und Maschinenbefehlen auch einen Abschnitt über den Programmstart von der Workbench bzw. dem CLI. Neben einer universellen Startup-Sequenz wird vor allen Dingen auf den Aufbau der Info-Datei eingegangen.

In der Hoffnung, daß Ihnen dieses Buch ein für allemal die Angst vor der Maschinensprache nimmt (und daß sich der Guru im Winterschlaf befindet...), kann ich Ihnen jetzt nur noch viel Spaß beim Programmieren wünschen.

Frank Riemenschneider

Der Umgang mit dem Devpac-Assembler

Einleitung

Wenn Sie mit diesem Buch am Computer arbeiten, werden Sie wahrscheinlich den Devpac-Assembler verwenden. Damit Sie aber bei der täglichen Arbeit nicht ständig zwischen diesem Buch und der Anleitung des Devpac hin- und herwechseln müssen, ist an dieser Stelle ein kleines Referenz-Kapitel eingefügt. Es erläutert Ihnen in Kürze die Kommandos und Optionen, die Ihnen das Programmpaket bietet.

Der Programmumfang

Die gelieferten Disketten beinhalten – neben den Include-Dateien und Libraries – einen Editor, einen Assembler, einen Linker sowie einen Debugger.

Sie können alle Programme getrennt verwenden. Um jedoch eine möglichst kurze Programmentwicklungszeit zu gewährleisten, sollten Sie das Angebot des Devpac in Anspruch nehmen, gleichzeitig mit dem Editor auch den Assembler und den Debugger im Speicher zu haben. Das bedeutet für Sie praktisch, daß Sie nach der Programmerstellung nicht erst umständlich den Editor verlassen, den Assembler aufrufen und anschließend den Debugger laden müssen. Vielmehr können Sie in dieser Umgebung den Assembler und den Debugger per Tastendruck aus dem Editor heraus aufrufen.

Es dürfte das einfachste sein, das an einem kurzen Beispiel einmal zu demonstrieren: Wenn Sie den Computer mit der ersten Diskette des Devpac gestartet haben und sich jetzt im CLI befinden, so geben Sie hier »genam2« ein. Sobald der Editor geladen wurde, können Sie ein Programm erstellen bzw. laden. Legen Sie jetzt die Diskette zu diesem Buch in ein Laufwerk und gehen Sie im Menü »Projekt« auf »Laden«. Sie erreichen diesen Menüpunkt auch über das Tastenkürzel [rechte Amiga-Taste]+[L]. Dort sollten Sie das entsprechende Laufwerk (df0: oder df1:), gefolgt von »Quelltexte/chart-grafik.quell«, eingeben. Nach einem abschließenden [Return] oder einem Mausklick auf »OK« wird das File geladen.

Wenn Sie diesen Quelltext assemblieren wollen, drücken Sie [A]+[A]. In Zukunft werde ich die [rechte Amiga-Taste] als [A] abkürzen. Diese Abkürzung gilt natürlich nur für die erste Taste einer solchen Kombination, das heißt, [Ctrl]+[A] kann nicht bedeuten, daß Sie die Control- und die rechte Amiga-Taste gleichzeitig drücken sollen. Entsprechend wird auch die Alt-Taste in Zukunft mit [Alt] abgekürzt werden.

Auf dem Bildschirm erscheint ein Menü. Die Bedeutung der einzelnen Felder wird später erläutert. Aber Sie sehen, man kann den Assembler ganz einfach aus dem Editor heraus starten, ohne diesen verlassen zu müssen. Genauso einfach ist es mit dem Debugger, den Sie mittels [A]+[D] aufrufen können, sofern Sie zuvor ein Programm »in den Speicher assembliert« haben. Was das genau bedeutet, erfahren Sie bei den Assembler-Optionen.

Das war einmal etwas Grundsätzliches. Jetzt beschäftigen wir uns mit den einzelnen Komponenten:

Der Editor

Die Funktionen des Editors können Sie über die Menüleiste erreichen. Sie sehen neben jedem Menüpunkt das entsprechende Tastenkürzel. Wenn Sie sich einmal an diese Tastenkürzel gewöhnt haben, werden Sie sie nicht mehr missen wollen, da Sie dadurch wesentlich schneller eine Funktion aufrufen können als über die Maus.

Da bleibt mir nur noch die Tastenbelegung für die Cursorbewegungen zu erläutern. Mit den vier Cursortasten können Sie wie gewohnt im Text herumfahren. Bei sehr langen Zeilen wird der Bildschirminhalt gescrollt, so daß Sie immer den Cursor sehen.Wenn Sie mit [↑] die erste Zeile des Windows erreicht haben und oben noch weiterer Text existiert, rollt der Text nach unten. Andernfalls erscheint die Meldung »Dateianfang«. Entsprechendes gilt für die unterste Window-Zeile. Nur scrollt hier der Text nach oben, oder es erscheint die Meldung »Dateiende«.

Für WordStar-Benutzer dürfte es angenehm sein, daß Sie im Editor den Cursor auch noch anders bewegen können, nämlich mit [Ctrl]+[S], [Ctrl]+[D], [Ctrl]+[E], [Ctrl]+[X].

Um den Cursor an den Anfang bzw. an das Ende einer Zeile zu bewegen, verwenden Sie [Ctrl] und die entsprechende Cursortaste ([→], [←]).

Um lediglich ein Wort weiterzuspringen, kombinieren Sie [Shift] mit den Tasten [→] bzw. [←]. Allerdings können Sie auch die WordStar-Kombinationen [Ctrl]+[A] bzw. [Ctrl]+[F] verwenden.

Seitenweise springen können Sie: nach oben mit [Shift]+[↑] oder [Ctrl]+[R], nach unten mit [Shift]+ ↓) bzw. [Ctrl]+[C].

Um ganz an den Anfang zu springen, drücken Sie [A]+[T] (können Sie dem Optionen-Menü entnehmen) oder [Alt]+[↑]. Das Dateiende erreichen Sie entweder mit [A]+[B] oder mit [Alt]+[↓].

Die Funktionen der [Del]- sowie der [Backspace]-Taste dürften allgemein bekannt sein. Zu [Tab] wäre noch zu sagen, daß es den ASCII-Code 9 benutzt. Es justiert den Cursor jeweils auf einer Spalte, die ein Vielfaches von 8 ist. Mit dieser Taste können Sie bequem Einrückungen vornehmen und Ihr Programm somit übersichtlicher gestalten.

Weitere Tastaturkommandos betreffen das Löschen von Textstellen.

So können Sie mit [Ctrl]+[Y] die aktuelle Zeile ganz löschen, während [Ctrl]+[Q] von der Cursor-Position bis zum Zeilenende löscht (entspricht [Ctrl]+[Q]+[Y] in WordStar).

Sollten Sie aus Versehen eine Zeile gelöscht haben, wird sie mit [Ctrl]+[U] wieder eingefügt. Allerdings können Sie damit auch eine Zeile mehrmals hintereinander kopieren.

Wie Sie den kompletten Text löschen, können Sie dem Menü »Datei« entnehmen: mit [A]+[C].

Zu guter Letzt müssen wir noch auf die Block-Kommandos eingehen, die alle auf die Funktionstasten gelegt sind. Den Anfang eines Blockes markieren Sie mit [F1], das Ende mit [F2]. Den Block können Sie nun speichern (mit [F3]), kopieren (Cursor an eine andere Stelle bewegen und dort [F4] drücken), löschen ([Shift]+[F3] oder [Shift]+[F5]), in den Blockspeicher kopieren ([Shift]+[F4]), aus dem Blockspeicher an der jetzigen Cursorstelle einfügen ([F5]) oder aber ihn ausdrucken ([A]+[W], siehe auch Datei-Menü).

Die Funktionstastenbelegung sowie den freien Speicherplatz und die aktuelle Devpac-Version erfahren Sie auch mittels der Tastenkombination [A]+[H] (oder im Menü Programm unter dem Punkt »Hilfe«).

Mit dieser Beschreibung sollten Sie den Editor beherrschen. Natürlich bedarf es einiger Übung, bis Sie mit allen Funktionen vertraut sind. Einen letzten Punkt habe ich Ihnen bis jetzt aber noch verschwiegen: Sie können ein paar Voreinstellungen ändern: Bewegen Sie dazu bitte die Maus in das Menü »Optionen« und hier auf den Punkt »Voreinstellungen«.

In dem neuen Requester können Sie einiges ändern. So z.B., wie weit die Tabulatoren-Stops auseinander liegen sollen und wie groß der Textpuffer gewählt werden soll. Bei dieser zweiten Möglichkeit können Sie einen beliebigen Wert zwischen vier und 999 eingeben. Allerdings sollten Sie darauf achten, daß noch genügend Platz (u.a. für den Debugger) übrigbleibt.

Die nächste Option (»Backups«) sorgt dafür, daß die alte Programmversion noch einmal gespeichert wird, auch wenn Sie zwischenzeitlich einige Änderungen daran vorgenommen haben. Dies ist eine einfache Methode, um vermeintliche Verbesserungen wieder rückgängig zu machen.

Ich deutete oben bereits an, daß Programme mit Einrückungen übersichtlicher gestaltet werden können. Um Ihnen bei diesen Bemühungen zu helfen, können Sie das »automatische Einrücken« verwenden. Es besagt lediglich, daß eine Folgezeile genausoweit eingerückt wird wie eine Zeile, die Sie mit [Return] abgeschlossen haben.

Wenn Sie bei »Zeilenende« auf »Umbruch« klicken, können Sie am Zeilenende mit [→] in die nächste Zeile bzw. am Zeilenanfang mit [←] in die obere Zeile fahren. Bei »Stop« dagegen bleibt der Cursor am Zeilenanfang stehen und scrollt beim Zeilenende einfach weiter.

Wie ich Ihnen bereits weiter oben erläuterte, können Sie den Assembler und den Debugger vom Editor aus aufrufen. Bei »MonAm laden« können Sie allerdings bestimmen, ob Sie

den Debugger im Speicher haben wollen. Wenn Sie hier »Nein« anklicken, müssen Sie erst den Editor verlassen, bevor Sie den Debugger aufrufen und laden können. Allerdings sparen Sie dadurch 24 Kbyte Speicher. Wenn Sie diese Option ändern und speichern, wird sie allerdings erst beim nächsten Start des Editors wirksam.

Wenn Sie möchten, daß GenAm den kompletten Bildschirm ausnutzt, klicken Sie bei »Volle Größe (aut.)« auf »Ja«.

Schließlich können Sie diese Voreinstellungen mit »Sichern« auf Diskette speichern. Beachten Sie aber bitte, welches Verzeichnis auf der Diskette gerade aktiv ist! Beim nächsten Starten versucht GenAm nämlich aus dem dann aktiven Verzeichnis seine Informationsdatei (»GENAM2.INF«) zu lesen. Falls nötig, können Sie die Datei noch in ein anderes Verzeichnis legen.

Der Assembler

Zunächst müssen wir unterscheiden, wie Sie den Assembler aufrufen: entweder mittels Tastendruck aus dem Editor oder vom CLI aus (»genim2«).

Nehmen wir an, Sie starten ihn vom Editor aus: Zunächst öffnet sich ein Window, in dem Sie ein paar Voreinstellungen treffen können:

Programmtyp: Wählen Sie »Ausführbar«, so wird ein Programm erstellt, das ohne weitere Vorbereitungen lauffähig ist (vorausgesetzt, es enthält keine Fehler). Bei »Linkable« müssen Sie vor dem Starten noch den Linker aufrufen, um benötigte Dateien einzubinden. Wie das genau funktioniert, werden wir später noch behandeln.

Bei »Groß/Klein« können Sie wählen, ob große und kleine Buchstaben unterschiedlich behandelt werden sollen. Bei »Verschieden« gelten die Namen "kleinundGROSS" und "kleinundgross" als zwei völlig unterschiedliche Bezeichnungen, während sie im anderen Fall als gleich betrachtet werden.

Sie können Ihrem Programm »Debug-Informationen« anhängen. Das bedeutet, daß der Debugger Bezeichnungen, die Sie in Ihrem Programm verwenden, beim Disassemblieren anzeigt. Das erleichtert Ihnen die Arbeit ungemein. Die Funktion »Export.« erlaubt es Ihnen sogar, 22stellige Bezeichner anstelle der sonst üblichen 8stelligen zu verwenden.

Während des Assembliervorgangs können Sie sich ein Listing ausgeben lassen. Ich denke, die verschiedenen Optionen sind klar.

Normalerweise sollten Sie das »Schnelle Assemblieren« anklicken. Sollte jedoch der Speicherplatz knapp werden, weicht der Assembler bei »Langsam« so oft wie möglich auf Diskette aus.

Bei »Output« schließlich können Sie wählen, was aus Ihrem Programm gemacht wird. »Nein« bedeutet einfach, daß nicht assembliert, sondern nur ein Syntaxcheck durchgeführt wird, während bei »Speicher« das Programm in den Speicher assembliert wird. Der Debugger kann auf diese Version zugreifen, wenn er mit [A]+[D] aufgerufen wird. Schließ-

lich und endlich können Sie das fertige Programm aber auch auf Diskette schreiben lassen. Geben Sie dazu einen Filenamen und u.U. auch den nötigen Pfad an. Vergessen Sie trotzdem nicht, das Wort »Diskette« anzuklicken !

Der Aufruf des Assemblers vom CLI aus ist denkbar einfach: »genim2 Datei [-Optionen]. Die Optionen sind im folgenden tabellarisch aufgeführt (sie sollten bei der Eingabe mit einem "–"-Zeichen beginnen):

B keine Binärdatei erzeugen (entspricht dem Syntaxcheck)
C Groß-/Kleinschreibung identisch (OPT C-)
D Debug (OPT D+)
L linkbarer Code (OPT L+)
M langsamer Assembliermodus
O Name der erzeugten Datei folgt dem O ohne Leerzeichen
P Name der Listing-Datei folgt dem P ohne Leerzeichen
Q Am Ende auf einen Tastendruck warten
T Tab-Abstand
X Erweiterter Debug (OPT X+)

Dies sind die Optionen der Version 2. Sie unterscheiden sich in einigen Punkten von der alten Version!

Hin und wieder wurde in Klammern (OPT ...) angegeben. Diese Angaben können im Quelltext erscheinen und bewirken dort dasselbe wie die hier aufgeführten Optionen (soweit angegeben). Auf diese zweite Möglichkeit werden wir später noch einmal zu sprechen kommen und sie dann ausführlich erläutern.

Hier noch ein Beispiel, um Ihnen die Verwendung der Optionen zu verdeutlichen:

genim2 test -ldpprt: Dieser Befehl assembliert das File test.s in ein linkbares Format mit vollständiger Debug-Information und schickt ein Listing an den Drucker.

Wenn Sie bei dem Dateinamen keine Erweiterung (wie .PRG) angeben, geht der Assembler von der Extension .S aus. Bei dem kreierten Programm ist die Erweiterung auch festgelegt: Wenn Sie einen Dateinamen explizit festlegen (mit –O), erhält die erzeugte Datei diesen Namen. Verwenden Sie nicht die –O-Direktive, ist der Name der des Quelltextes plus der Erweiterung .PRD, .BIN oder .O. Sollten Sie über die –O-Anweisung eine Erweiterung angeben, wird diese an den Namen des Quelltextes angehängt. Zu guter Letzt wird der Name des Quelltextes für die erzeugte Datei verwendet, wenn Sie gar nichts angegeben haben.

Noch ein Wort zu den unterschiedlichen Extensions: .O oder .OBJ-Dateien müssen noch gelinkt werden, bevor sie ablaufen, Programme ohne Erweiterung können direkt ausgeführt werden.

Beachten Sie bitte eines: Es ist ein Unterschied, ob ein Programm von der Workbench aus aufgerufen wird oder vom CLI aus! Lesen Sie dazu bitte auch Anhang A einmal durch.

Jetzt kommen wir zu den Kommandos, die Sie im Quelltext angeben können, und die den Assembler kontrollieren.

END
bezeichnet das Ende des Quelltextes; alles, was danach kommt, wird ignoriert. Daher sollte diese Anweisung am besten in der letzten Zeile stehen.

INCLUDE Dateiname
Mit dieser Anweisung wird von der Diskette ein Quelltext geladen und an der Stelle in Ihr Programm eingefügt. Dabei muß der Dateiname im gültigen Amiga-Dos-Format angegeben sein.

Dadurch können Sie sich eine Menge Tipparbeit sparen. Z.B. ist diese Routine aus Anhang A auf der Diskette zum Buch im Directory »include« gespeichert und kann von dort aus sehr einfach über die INCLUDE-Anweisung geladen werden. Auch der Devpac-Assembler bietet von sich aus eine ganz ähnliche Routine; allerdings befindet sich die auf der Diskette 1 im Verzeichnis "include/misc/" und heißt "easystart.i".

INCDIR "Directory"
Dadurch können Sie generell angeben, in welchem Verzeichnis nach den Include-Dateien gesucht werden soll. Dadurch können Sie wiederum viel Tipperei sparen, da Sie nicht bei jedem Include-File den kompletten Pfad mitangeben müssen. Sie können auch mehrere Directories (durch Kommas getrennt) angeben.

INCBIN Dateiname
Diese Direktive lädt keinen Quelltext, sondern eine binäre Datei in Ihre Programme. Dadurch können Sie z.B. Bilder verwenden, ohne sie umständlich in DC-Direktiven umwandeln zu müssen.

OPT Option
Mit dieser Direktive können Sie mehrere Optionen (wieder durch Kommas getrennt) setzen. Welche Optionen Ihnen zur Verfügung stehen, folgt in einer kurzen Zusammenstellung:

OPT A
Bei OPT A+ wird die PC-relative Adressierung verwendet, wo immer möglich.

Durch den Gebrauch dieser Option können die Programmgröße und die Laufzeit erheblich reduziert werden. Das garantiert aber nicht, daß positionsunabhängiger Code vorliegt!

OPT C
Bei C– wird nicht zwischen Groß- und Kleinschreibung unterschieden. Mit C+ kann das wieder rückgängig gemacht werden.

OPT D
An das File-Ende wird eine Symboltabelle gehängt, die der Debugger MonAm lesen kann. Die Option wird mit D+ eingeschaltet. Wenn mehrere OPT D-Anweisungen vorkommen, gilt nur die letzte.

OPT L
Normalerweise erzeugt der Assembler ausführbaren Code. Mit L+ wird aber linkbarer Code erzeugt. Diese Anweisung muß in der ersten Zeile stehen (auch keine Leerzeile davor !).

OPT M
Wenn der Assembler ein Listing erzeugt, wird – wie im Quelltext – nur der Aufruf eines Makros erzeugt. Mit OPT M+ kann aber an dieser Stelle das komplette Makro (also alle enthaltenen Befehle) angezeigt werden. Sie können diese Direktive so oft ein- und ausschalten wie Sie wollen.

OPT N
Falls das Listing für Ihren Drucker zu breit werden sollte, können Sie N+ setzen. Bei der Ausgabe werden dann Zeilennummern und Objekt-Code nicht mit ausgegeben.

OPT O
Hiermit können Sie den Assembler veranlassen, Ihre Programme zu optimieren. Es gibt verschiedene Stufen:

O1+ optimiert Rückwärtssprünge

O2+ optimiert Adreßregister indirekt mit Offset in Adreßregister

O+ schaltet alle Optimierungen ein

O – schaltet alle Optimierungen aus

OW – schaltet die bei Optimierung ausgegebenen Warnungen aus

Mit dem entgegengesetzten »Nachzeichen« +/– erreichen Sie jeweils die andere Einstellung.

OPT P
GenAm erlaubt normalerweise positionsunabhängigen Code. Wenn Sie nur PC-relativen Code erzeugen wollen, können Sie P+ als Kontrolle verwenden. Falls das Programm nicht positionsunabhängig ist, wird ein Fehler ausgegeben. Mit P– kann die Kontrolle wieder ausgeschaltet werden; beide Anweisungen können in einem Programm mehrmals gebraucht werden, um sicherzustellen, daß bestimmte Teile unabhängig sind.

OPT S
Falls ein Listing ausgegeben wird, erscheint am Ende auch eine Symboltabelle. Die Ausgabe dieser Tabelle können Sie mit S– ausschalten. Dies ist ratsam, wenn Sie include-Files verwenden, da die Symboltabelle in einem solchen Fall sehr lang werden kann.

OPT T
GenAm überprüft, ob Ausdrücke absolut oder relativ sind; dadurch kann er Programmierfehler erkennen. Da diese Überprüfungen bei manchen Anwendungen hinderlich sind, können sie mit T– ausgeschaltet werden.

OPT U
Normalerweise beginnen lokale Labels mit einem Punkt. Damit sie aber auch mit einem Unterstrich »_« beginnen können, muß die Option U+ gesetzt werden.

OPT W
Mit W– schalten Sie die Warnmeldungen des Assemblers aus.

OPT X
Dies ist eine erweiterte Version der Option D. Hiermit werden Symbole mit bis zu 22 Zeichen Länge anstelle der üblichen 8 angegeben.

Aber es gibt noch mehr Direktiven:

<LABEL> EVEN
Diese Anweisung garantiert, daß der nächste Befehl auf eine Wortgrenze justiert wird. Dieser Befehl ist nicht oft nötig, da GenAm alle Befehle von selbst auf Wortgrenzen legt (Ausnahme: DC.B und DS.B).

CNOP Offset,Justierung
Die Direktive justiert den Programmzähler nach dem Offset und der Justierungsart. Eine Justierungsart von 2 heißt Wort-, eine von 4 bedeutet Langwortgrenze.

Beispiel: CNOP 1,4 setzt den Programmzähler auf die nächste Langwortgrenze plus 1 Byte. Hin und wieder benötigt man beim Betriebssystem eine Langwortjustierung, die man mit CNOP 0,4 erreicht. Der Programmstart wird immer langwortjustiert.

<LABEL> DC.B Ausdruck<,Ausdruck> ...
<LABEL> DC.W Ausdruck<,Ausdruck> ...
<LABEL> DC.L Ausdruck<,Ausdruck> ...
Diese Anweisungen definieren Konstanten im Speicher. Dabei können mehrere Ausdrücke durch Kommas getrennt angegeben werden. Allerdings können in einer solchen Anweisung nicht mehr als 128 Byte erzeugt werden. Beachten Sie bitte, daß nach dem ersten Ausdruck keine Leerstellen kommen, da eine Leerstelle den Kommentar einleitet. So erzeugt die Anweisung »dc.b 1,2,3, 4« lediglich 3 Byte !

<LABEL> DS.B Ausdruck
<LABEL> DS.W Ausdruck
<LABEL> DS.L Ausdruck
Mit dieser Anweisung kann Speicherplatz reserviert werden, dessen Inhalte auf Null gesetzt werden.

<LABEL> DCB.B Zahl,Wert
<LABEL> DCB.W Zahl,Wert
<LABEL> DCB.L Zahl,Wert
Damit können aus der spezifizierten Größe konstante Blöcke von Daten erzeugt werden. Dabei zeigt »Zahl« an, wie oft »Wert« wiederholt werden soll.

FAIL
Diese Anweisung gibt den Fehler "user error" aus. Damit kann der Programmierer gewarnt werden, wenn er eine falsche Anzahl von Parametern an ein Makro übergeben will.

OUTPUT Dateiname
Damit können Sie den Namen der Ausgabedatei bestimmen. Falls dieser Name mit einem Punkt beginnt, wird nur die Erweiterung beeinflußt. Der Name wird ansonsten nach den oben beschriebenen Regeln gebildet.

_G2
Damit kann überprüft werden, mit welcher Version des Assemblers gearbeitet wird. Das Symbol enthält die Version des Assemblers und ist absolut.

<LABEL> REPT Ausdruck
ENDR
Mit diesen Anweisungen können mehrere Zeilen wiederholt werden. Wie oft die Schleife durchlaufen werden soll, ist in »Ausdruck« definiert. Ein Beispiel:

```
REPT 512/4
move.l (a0)+,(a1)+
ENDR
```

Beachten Sie bitte, daß Sie innerhalb der Schleife keine Labels definieren, da sonst die Meldung »label defined twice« erscheint.

LIST
Nach dieser Direktive wird in Pass 2 ein Listing auf das vorgewählte Gerät ausgegeben.

NOLIST
schaltet das Listing wieder aus.

PLEN Ausdruck
Damit wird die Seitenlänge in Zeilen auf den Wert von »Ausdruck« gesetzt. Der Wert muß zwischen 12 und 255 liegen, voreingestellt sind 60.

LLEN Ausdruck
Die Zeilenlänge kann damit bestimmt werden. Voreingestellt ist 132, der neue Wert muß zwischen 38 und 255 liegen.

TTL String
Auf jeder neuen Seite wird »String« als Titel ausgedruckt. Falls kein Titel vereinbart wurde, wird der aktuelle Include-Dateiname verwendet. Sollte der String Leerstellen enthalten, muß er in Hochkommas gesetzt werden.

SUBTTL String
Damit kann ein Untertitel festgesetzt werden. Die erste Direktive gilt für die erste Seite usw.

SPC Ausdruck
erzeugt so viele Leerzeilen im Ausdruck, wie »Ausdruck« angibt.

PAGE
bewirkt einen Seitenvorschub im Listing.

LISTCHAR Ausdruck<,Ausdruck>
Damit kann eine angegebene Zeichenfolge an das Ausgabegerät (ausgenommen den Bildschirm) gesendet werden. Wird die Ausgabe z.B. an das Gerät prt: gesendet, kann mit listchar 27,'[','4','w' die Schmalschrift eingeschaltet werden. Bei den Schnittstellen par: und ser: müssen Sie den druckereigenen Code senden.

Kommen wir jetzt zu den Label-Direktiven.

Label EQU Ausdruck
EQUate heißt soviel wie Gleichsetzung; damit werden Wert und Typ von »Label« dem Ergebnis von »Ausdruck« gleichgesetzt. Da einige andere Assembler für diese Funktion das Gleichheitszeichen verwenden, wurde dies auch für den DevPac-Assembler übernommen. Damit sind

Label EQU Ausdruck und
Label = Ausdruck identisch.

Label EQUR Register
Somit können Sie ein Daten- oder Adreßregister mit einem von Ihnen festgesetzten Namen ansprechen. Das wird als »Registerequate« bezeichnet. Es muß vor dem ersten Gebrauch definiert werden.

Label SET Ausdruck
Damit können Sie einem Label immer wieder neue Werte zuweisen. Vorwärts-Referenzen sind nicht zulässig. Mit dieser Anweisung kann man Zähler in Makros leicht realisieren:

zcount set zcount+1

Dabei wird allerdings vorausgesetzt, daß zcount vorher auf Null gesetzt wurde.

Label REG Registerliste
Mit der REG-Direktive können Sie einem Symbol eine Registerliste zur Verwendung mit MOVEM zuordnen. Somit wird die Gefahr verringert, am Anfang und Ende einer Routine unterschiedliche Registerlisten zu haben. Mit REG definierte Symbole können ausschließlich mit MOVEM benutzt werden.

<LABEL> RS.B Ausdruck
<LABEL> RS.W Ausdruck
<LABEL> RS.L Ausdruck
Mit diesen Anweisungen können Sie eine Liste von konstanten Labels erstellen, womit Sie Datenstrukturen und globale Variablen leichter handhaben können. Um dies besser erläutern zu können, möchte ich ein kurzes Beispiel anführen:

```
onstate   rs.b 1
start     rs.l 1
end       rs.l 1
```

Sie können dann folgendermaßen auf sie zugreifen:

```
move.b onstate(a6),d1
move.l start(a6),d0
cmp.l end(a6),d0
```

Jede dieser Direktiven benutzt einen eigenen Zähler, der bei Beginn eines jeden Passes auf Null gesetzt wird. Trifft der Assembler auf eine solche Direktive, wird der Zähler gemäß Größe und Mächtigkeit der Direktive erhöht. Wenn die obigen Definitionen die ersten RS-Direktiven wären, stünde »onstate« auf Null, »start« auf 2 und »end« auf 6.

RSRESET
Der interne Zähler wird zurückgesetzt, wie bei der RS-Direktive.

RSSET
Damit kann man den Zähler auf einen bestimmten Wert setzen.

RS
Dieses reservierte Symbol beinhaltet den aktuellen Wert des Zählers.

Nach den Label-Direktiven folgen die Anweisungen für bedingtes Assemblieren. Diese Methode erlaubt es,

- ein Programm für zwei verschiedene Maschinen zu schreiben,
- Debug-Codes für die Testversion einzufügen.
- Zwei leicht unterschiedliche Versionen eines Programms werden benötigt.

Die Entwickler des DevPac haben nach eigenen Angaben diese Methode verwendet, da der Assembler auf zwei Maschinen läuft, und es nur in der internen Version des GenAm Debug-Codes gibt.

Die Assemblier-Bedingungen können durch das Benutzen von Argumenten (in Makros) und durch die Definition von Symbolen mittels EQU und SEt-Anweisungen festgelegt werden. Zu Beginn einer solchen Bedingung muß eine der möglichen IF-Direktiven stehen, am Ende ein ENDC. Solche Blöcke können bis zu einer Tiefe von 65535 geschachtelt werden.

Folgende IF-Ausdrücke sind zulässig:

IFEQ Ausdruck
IFNE Ausdruck
IFGT Ausdruck
IFGE Ausdruck
IFLT Ausdruck
IFLE Ausdruck

Diese Anweisungen entwickeln den Ausdruck, vergleichen das Ergebnis mit Null und schalten dementsprechend das Assemblieren an oder aus. Dabei entsprechen die Bedingungen den Codes des 68000. Ein kleines Beispiel soll das verdeutlichen:

```
         IFEQ DEBUG
logon        dc.b 'Kommando eingeben: ',0
         ENDC
         IFNE DEBUG
         opt d+
logon        dc.b 'Wird's bald ?',0
         ENDC
```

Angenommen, DEBUG hat den Wert 1, schaltet die erste Bedingung das Assemblieren aus, weil 1 nicht gleich (EQal) Null ist; die zweite Bedingung schaltet das Assemblieren ein, weil 1 nicht gleich (NotEqual) Null ist.

IFD Label
IFND Label
Diese beiden Direktiven erlauben die Kontrolle in Abhängigkeit davon, ob ein Label bereits definiert ist oder nicht. IFD schaltet das Assemblieren ein, falls das Label schon existiert; IFND bewirkt das Gegenteil.

IFC 'String1','String2'
IFNC 'String1','String2'
Mit diesen Anweisungen kann getestet werden, ob zwei Strings gleich sind; dementsprechend wird die Assemblierung ein- bzw. ausgeschaltet. Dieser Test ist z.B. sinnvoll, wenn eines der beiden Argumente (oder beide) Parameter sind, die bei einem Makro-Aufruf verwendet werden.

ELSEIF
schaltet das bedingte Assemblieren an oder aus.

ENDC
beendet die aktuelle IF-Ebene. Wenn es mehr IF- als ENDC-Anweisungen gibt, wird eine Fehlermeldung ausgegeben.

IIF Ausdruck Anweisung
Dabei handelt es sich um eine Kurzform der IFNE-Direktive, die das bedingte Assemblieren einer Anweisung oder einer Direktiven ermöglicht. Hierbei darf eine ENDC-Anweisung nicht gesetzt werden.

Kommen wir nun zu den Makro-Operationen.

Label MACRO
Damit beginnt eine Makro-Operation; dies veranlaßt GenAm, alle folgenden Zeilen in den Makro-Puffer zu kopieren, bis ein ENDM auftritt. Makros können im übrigen geschachtelt werden.

ENDM
beendet ein Makro. In dieser Zeile darf kein Label stehen.

MEXIT
Mit dieser Anweisung können Sie ein Makro vorzeitig verlassen.

NARG
NARG enthält als Wert die Anzahl der Parameter, die an das Makro übergeben wurden. Es handelt sich dabei um ein reserviertes Symbol. NARG ist eines der wenigen Symbole, die wirklich groß geschrieben werden müssen.

Innerhalb eines Makros können Sie mit dem Zeichen Backslash (»\«), gefolgt von einer Zahl (1 bis 9), auf die übergebenen Parameter zugreifen. Der Parameter »\0« bewirkt, daß der Typ (B, W, L) beibehalten wird, der dem Makro beim Aufruf angehängt wurde (fehlt der Typ, wird W angenommen). Soll bei jedem Makro-Aufruf ein unterschiedliches Label erzeugt werden, ist die Direktive »\@« zu verwenden.

Damit wären die Direktiven des Makro-Assemblers GenAm weitestgehend erläutert. Allerdings habe ich bereits angedeutet, daß man auch Programme erzeugen kann, die erst noch gelinkt werden müssen. Dies wird man nur dann tun, wenn man Kompilate von Hochsprachencompilern (wie Modula-2, C) mit Assemblerprogrammen kombinieren will, oder wenn man Programme in einem Team entwickelt.

Schreiben Sie dagegen allein Programme nur in Assembler, ist es für Sie das beste, die Programme ausführbar in den Speicher zu assemblieren. Das hält die Assemblierzeiten gering, die Linkzeiten entfallen völlig, und Sie können den Debugger MonAm sehr komfortabel handhaben.

Um alle Direktiven behandelt zu haben, müssen wir nun noch auf die eingehen, die mit den unterschiedlichen Ausgabedateien zusammenhängen.

SECTION String<,Typ>
Ein Programm kann aus mehreren Sektionen bestehen, die im endgültigen Programm mit gleichnamigen Sektionen zusammengefügt werden. »String« und »Typ« werden als Sektionsnamen verwendet. Sie können jeweils bis zu 32 Zeichen lang sein, dürfen nicht in Klammern stehen und keine Leerzeichen enthalten. Die Direktive verursacht einen Wechsel zu der genannten Sektion, der Typ wird optional definiert.

Es gibt folgende Typen:

CODE Codesektion, Public Memory
Code_F Codesektion, Fast Memory
Code_C Codesektion, Chip Memory
DATA Datasektion, Public Memory
DATA_F Datasektion, Fast Memory
DATA_C Datasektion, Chip Memory
BSS BSS-Sektion, Public Memory
BSS_F BSS-Sektion, Fast Memory
BSS_C BCC-Sektion, Chip Memory

Code-Sektionen benötigt man für ausführbare Anweisungen, DATA-Sektionen für initialisierte und BSS für nichtinitialisierte Daten. BSS-Sektionen haben den Vorteil, daß sie keinen Speicherplatz verbrauchen – es wird lediglich ihre Länge gespeichert.

Hinweis: Wenn Sie den Linker ALink in einer früheren Version als 2.3 benutzen, verwenden Sie keine Typen, die FastMemory oder ChipMemory erfordern, da es sonst zum Absturz kommt. (Bei DevPac V2 wird der verbesserte Linker BLink mitgeliefert.)

IDNT string
Diese Anweisung setzt den Namen »string« als Modulnamen ein. Für diesen Namen gilt dasselbe wie für »String« bei SECTION; je Modul ist nur eine solche Anweisung möglich.

Soll ein Programm gelinkt werden, so ist es wichtig, Symbole importieren und exportieren zu können. Der Assembler kann dabei relative von konstanten Symbolen (im Gegensatz zu Amiga-DOS) unterscheiden und somit helfen, Programmierfehler zu entdecken.

XDEF Export<,Export>...
Damit werden Labels für den Export in andere Programme definiert. Lokale Labels können nicht exportiert werden. Beim ausführbaren Datei-Format wird diese Direktive ignoriert.

XREF Import<,Import>...
XREF.L Import<,Import>...
Dadurch werden Labels als externe Referenzen definiert, d.h. sie werden erst im Linkerlauf von anderen Programmen oder Modulen importiert und sind im Assemblerlauf unbekannt. XREF.L definiert absolute Labels (=Konstanten), während XREF relative Labels (Programmadressen) meint.

Bei ausführbaren Dateien wird diese Direktive ignoriert.

COMMENT Kommentar
Diese Direktive wird grundsätzlich ignoriert.

ORG Ausdruck
Durch diese Anweisung wird der Assembler veranlaßt, positionsunabhängigen Code zu erzeugen; normalerweise benötigen Amiga-DOS-Programme jedoch nicht diese Anweisung. Diese Direktive sollte nur mit äußerster Vorsicht gehandhabt werden, da am Anfang der Datei zwar der Amiga-DOS-Header steht, am Ende aber die Relozierinformationen fehlen; daher wird das Programm wahrscheinlich nicht problemlos laufen.

OFFSET <Ausdruck>
Die Code-Generierung wird damit in eine besondere, Assembler-interne Sektion umgeschaltet. Der Ausdruck (muß nicht angegeben werden) setzt den PC dieser Sektion. Es werden keine Daten auf Diskette geschrieben; innerhalb dieser Sektion ist nur die DS-Direktive erlaubt. Werden Labels in dieser Sektion definiert, sind sie absolut.

_LK
Dieses reservierte Symbol zeigt an, welche Code-Art erzeugt wird:

3 linkbar
4 ausführbar

Damit hätten wir den Assembler »genim2« erschlagen. Jetzt fehlt nur noch der symbolische Debugger, »MonAm«.

Der Debugger

Auch den Debugger kann man – wie schon den Assembler – entweder vom CLI oder vom Editor aus aufrufen.

Vom CLI aus können Sie ihn mit »genam« aufrufen. Nach dem Start wird ein Window geöffnet, das Sie nach dem Filenamen fragt. Diesen Filenamen können Sie auch auf der Befehlszeile im CLI bereits angeben, wenn Sie wollen. Falls Sie keine Kommandozeile wünschen, drücken Sie einfach [Return].

Vom Editor aus können Sie den Debugger auf zwei Arten aufrufen: mit [A]+[M] und mit [A]+[D]. Bei der ersten Tastenkombination müssen Sie – wie oben – den Filenamen eingeben. Bei der zweiten Tastenkombination wird ein in den Speicher assembliertes Programm verwendet; Voraussetzung dafür ist natürlich, daß Sie zuvor den Assembler mit einem Programm aufgerufen haben.

Sobald der Debugger Ihr Programm geladen hat, erscheint die Meldung: Exception: Breakpoint. Das heißt, daß der Debugger an die erste Programmzeile einen Breakpoint gesetzt hat und jetzt auf Ihre Eingaben wartet.

Was bedeutet eigentlich »symbolischer Debugger«? Wie weiter oben bereits mehrfach angedeutet, erscheinen im Assemblier-Fenster alle Labels, die Sie gesetzt haben. Dadurch können Sie sich in Ihrem Programm wesentlich besser zurechtfinden.

Auf dem Bildschirm sehen Sie drei Fenster. Nur eines davon kann jeweils aktiv sein. Das erkennen Sie daran, daß ein Titel revers dargestellt ist, während die beiden anderen Titel normal dargestellt sind.

Wenn Sie die [Tab]-Taste drücken, können Sie jeweils ein Fenster weiter springen und somit das nächste Fenster aktivieren. Doch was zeigen die Fenster eigentlich an ?

Fenster 1 zeigt alle Daten- und Adreßregister sowie den Speicher, auf den sie zeigen. Fenster 2 zeigt mehrere Zeilen des disassemblierten Codes an. Die Anfangsadresse des Windows ist normalerweise der PC. Fenster 3 zeigt einen Teil des Speichers in Hex- und ASCII-Darstellung. Der untere Bildschirmbereich wird für Meldungen freigehalten.

Ein Nachteil der [Tab]-Taste ist, daß man jeweils nur das nächste Fenster ansprechen kann. Wenn Sie beispielsweise von Fenster 2 zu Fenster 1 wechseln wollen, können Sie aber auch direkt [A]+[1] drücken. Entsprechendes gilt natürlich auch für die beiden anderen Fenster.

Kommen wir jetzt zu einer Übersicht über die Funktionen von MonAm:

[A]+[A] Adresse setzen
Mit diesem Befehl können Sie die Anfangsadresse eines Fensters setzen. Das bedeutet, daß die erste ausgegebene Zeile in diesem Window an der neuen Adresse liegt. So können Sie die Ausgabe z.B. im Speicherfenster ganz einfach festlegen.

[A]+[B] Breakpoint setzen
Mit diesem Befehl können Sie einen Breakpoint setzen.

[A]+[E] Editieren
Im Speicherfenster können Sie mit dieser Tastenkombination editieren. Mit den Cursortasten können Sie den Cursor im Fenster bewegen und mit den Tasten 0 bis 9 und A bis F Speicherbereiche ändern. Mit der [Tab]-Taste können Sie in diesem Fenster zwischen dem Hex- und dem ASCII-Editiermodus wählen. Verlassen können Sie das Fenster wieder mit [Esc].

Im Registerfenster bewirkt [A]+[E] dasselbe wie [A]+[R], nämlich das Setzen von Registern.

Im Source-Fenster können Sie mit [A]+[E] den Tab-Abstand zwischen 4 und 8 umschalten.

[A]+[L] Fenster binden
Mit diesem Befehl können Sie ein Fenster an ein anderes oder an ein Register binden. Nach einer Exception werden die Startadressen aller Fenster neu berechnet und ggf. verändert. Um eine Bindung zu löschen, geben Sie in die Dialogbox einen Leerstring ein.

Fenster 2 ist standardmäßig an den PC gebunden. Möchten Sie ein Fenster an ein anderes binden, müssen Sie dessen Speicher angeben. Auf diese Speicher können Sie mit M2, M3, M4 und M5 zugreifen (Sie können über bis zu fünf Fenster verfügen; doch darüber später mehr). Nur eines müssen Sie beachten: Falls Fenster 4 als Source-Fenster benutzt wird, weisen Sie ihm nie einen neuen Wert zu !

[A]+[O] Auswerten
In einer Dialogbox können Sie einen Ausdruck eingeben, der anschließend ausgewertet wird. Die Ausgabe erfolgt in Hex, Dezimal und ggf. als Symbol.

[A]+[P] Drucker-Dump
Der Inhalt des aktuellen Fensters wird auf dem Drucker ausgegeben. Sie können diese Funktion mit [Esc] abbrechen.

[A]+[R] Register setzen
Hiermit können Sie ein Register neu setzen. Dazu geben Sie das Register an, gefolgt von einem Gleichheitszeichen und einem Ausdruck (z.B.: A3=A2+4).

[A]+[S] Fenster spalten
Dieser Befehl splittet das Fenster 2 in die Fenster 2 und 4 bzw. das Fenster 4 in die Teile 3 und 5. Die beiden neuen Fenster sind vom ursprünglichen unabhängig. Wenn Sie noch einmal [A]+[S] drücken, werden zwei wieder zusammengefügt.

[A]+[T] Typ verändern
Damit können Sie Fenster 4 zwischen Disassemblierung, Speicheranzeige und Source umschalten. Allerdings funktioniert dieser Befehl auch, wenn man eine ASCII-Datei geladen hat.

[A]+[Z] Zoom
Das aktuelle Fenster wird mit diesem Befehl auf volle Bildschirmgröße gebracht. Mit [Esc] oder einem zweiten [A]+[Z] wird das Fenster wieder verkleinert.

Mit den Cursortasten können Sie die Startadresse eines Speicher- oder Disassemblierungsfensters verändern.

In einem Speicherfenster verändern alle Cursortasten die Startadresse des Fensters. Mit [Shift]+[↑] bzw. [Shift]+[↓] können Sie seitenweise nach oben/unten blättern.

In einem Disassemblierungs-Fenster verändern [↑] und [↓] die Startadresse jeweils um eine Instruktion, während [→] und [←] auf Wortbasis arbeiten.

Im Source-Fenster verändern [↑] und [↓] die Anzeigen zeilenweise, [Shift]+[↑] und [Shift]+[↓] seitenweise.

Während des Programmablaufs können Sie die Ausgabe betrachten; verwenden Sie dazu die System-Gadgets der Fenster. Wollen Sie den Workbench-Screen sehen, drücken Sie [linke AMIGA-Taste]+[N] und [linke AMIGA-Taste]+[M], um wieder zurückzukehren.

Kommen wir jetzt zu den Breakpoints, die Sie wahrscheinlich am meisten von den Funktionen des MonAm benötigen werden.

MonAm unterscheidet zwischen einfachen Breakpoints (das Programm wird dort angehalten, anschließend wird der Breakpoint gelöscht), Stop-Breakpoints (Breakpoint dort, wo ein Befehl zum n-ten Mal ausgeführt wird), Zähler-Breakpoint (an der Stelle wird ein Zähler erhöht, aber das PRG läuft weiter), permanente Breakpoints (wie ein einfacher Breakpoint, wird aber nicht gelöscht) und bedingte Breakpoints (je nach Bedingung).

[A]+[B] Breakpoint setzen
Mit diesem Befehl können Sie Breakpoints setzen und löschen. Ihre Eingabe muß allerdings einem bestimmten Format entsprechen:

`<Adresse>`
setzt einen einfachen Breakpoint.

`<Adresse>,<Ausdruck>`
setzt einen Stop-Breakpoint an der Adresse; das Programm wird angehalten, wenn <Ausdruck> eine bestimmte Anzahl Mal ausgeführt wurde.

`<Adresse>,=`
setzt einen Zähler-Breakpoint; der Zähler wird mit Null initialisiert.

`<Adresse>,`
setzt einen permanenten Breakpoint.

`<Adresse>,?<Ausdruck>`
setzt einen Breakpoint, der von Ausdruck abhängig ist.

`<Adresse>,-`
löscht an der angegebenen Adresse jeglichen Breakpoint.

Die Adressen dürfen nicht ungerade und nicht unlesbar sein; auch dürfen sie nicht im ROM gesetzt werden. Allerdings kann man diese Breakpoints (im ROM) mit dem Run-Until-Befehl verwenden.

Bei jedem Breakpoint wird der Prozessor-Status im History-Speicher festgehalten, gleichgültig, ob das Programm unterbrochen wird oder nicht.

Help Breakpoint- und Segmentanzeige
Der Befehl zeigt die Adressen der Text-, Daten- und BSS-Segmente sowie deren Länge mit allen gesetzten Breakpoints an.

[Ctrl]+[B] **Breakpoint setzen**
Dieser Befehl setzt einen einfachen Breakpoint an der Startadresse des aktuellen Fensters. Ist schon ein Breakpoint vorhanden, wird er gelöscht. Dieser Befehl existiert nur noch aus Kompatibilität zur ersten Version des Debugggers.

[U] **Go-Until**
Der Debugger fragt Sie nach der Adresse für einen einfachen Breakpoint, das Programm läuft aber anschließend weiter.

[Ctrl]+[K] **Kill Breakpoints**
Alle gesetzten Breakpoints werden gelöscht.

[Ctrl]+[A] **Breakpoint setzen und ausführen**
Dieser Befehl setzt einen Breakpoint auf eine Adresse hinter den PC. Somit können Sie am Ende einer Schleife (z.B. DBF-Schleife) einfach einen Breakpoint setzen und müssen dann nicht ständig diese Schleife durchlaufen.

[Ctrl]+[X] **Abbruch der Programmausführung**
Ihr Programm wird unterbrochen. Beachten Sie bitte, daß Sie manche DOS-Routinen nicht unterbrechen !

[H] **History-Speicher-Anzeige**
Dieser Puffer speichert maximal fünf Prozessor-Zustände (bei Breakpoints) und gibt sie mit dieser Funktion auf dem Bildschirm aus.

[Ctrl]+[C] **Abbruch**
Zunächst erscheint die Meldung »Programmende«. Wenn Sie noch einmal [Ctrl]+[C] betätigen, wird MonAm verlassen; anstelle dessen können Sie aber auch ein neues Programm laden.

Sollten Sie MonAm vom Editor aus aufgerufen haben, kommen Sie bereits mit dem ersten Tastendruck in den Editor zurück.

[Ctrl]+[Q] **Programm verlassen**
Damit verlassen Sie das zu debuggende Programm. Dabei wird allerdings ein zuvor belegter Speicherplatz nicht freigegeben !

[Ctrl] + [L] **Programm laden**
Damit können Sie ein ausführbares Programm laden. Sollte es sich um ein anderes Format handeln, müssen Sie das File als Binärdatei laden.

[B] **Binärdatei laden**
Bei der Startadresse müssen Sie keinen Wert eingeben. Das File wird dann automatisch an eine freie Stelle im Speicher geladen. Dabei enthält das Symbol M0 die Startadresse, M1 die Adresse des Dateiendes.

[S] **Binärdatei sichern**
Damit können Sie eine Binärdatei speichern. Sollten sich Anfangs- und Endadresse nicht geändert haben, können Sie bei den betreffenden Eingaben M0 und M1 eingeben.

[A] **ASCII-Datei laden**
Hiermit laden Sie eine ASCII-Datei. Falls Fenster 4 noch nicht geöffnet wurde, passiert das jetzt; es wird als Source-Fenster verwendet.

[Ctrl]+[R] **laufenlassen**
Das Programm läuft von der gegenwärtigen Adresse mit der normalen Geschwindigkeit ab. Normalerweise wird dieser Befehl nach einem Breakpoint verwendet.

[Ctrl]+[Y]/[Ctrl]+[Z] **Single-Step**
Der Maschinensprache-Befehl, auf den der PC zeigt, wird ausgeführt.

[Ctrl]+[T] **Befehl interpretieren**
Dieser Befehl interpretiert den Opcode am PC. Er ist dem Single-Step-Befehl ähnlich, verfolgt aber keine JSR-, BSR- Trap-, Line-A- oder Line-F-Befehle. Somit können Sie sich das Durchlaufen einer bereits getesteten Routine (z.B. in einer Library) ersparen.

[Ctrl]+[S] Befehl überspringen
Wenn Sie wissen, daß ein Befehl ungewollte Aktionen auslöst, sollten Sie mit dieser Instruktion den am PC liegenden Befehl überspringen, nicht mit [Ctrl]+[Z] ausführen lassen. Der PC wird dafür um die Größe der aktuellen Instruktion erhöht.

[R] Run
Diesen Befehl können Sie auf zwei Arten ausführen:

RUN G
ist identisch mit [Ctrl]+[R].

RUN I
Sie müssen einen Parameter eingeben, der besagt, wie viele Instruktionen ausgeführt werden, bevor MonAm wieder Ihr Programm stoppt.

[G] Speicher durchsuchen
Auf diesen Tastendruck hin erscheint die Meldung »Suchen nach B/W/L/T/I«, wobei die Kürzel für Byte, Wort, Langwort, Text und Instruktion stehen.

Bei B, W und L werden Sie nach einer Zahlensequenz gefragt. Dabei findet MonAm diese Zahlen auch an ungeraden Adressen. Bei T müssen Sie einen Text eingeben, nach dem gesucht werden soll. Beachten Sie bitte, daß MonAm bei der Suche zwischen Groß- und Kleinschreibung unterscheidet. Bei I können Sie einen Maschinensprachebefehl ganz oder teilweise eingeben (z.B. $14 (A findet u.a. den Befehl MOVE.L D2,$14(A0)).

Beachten Sie auch bitte das Ausgabeformat des Disassemblers: Verwenden Sie für Zahlen bitte Hex-Code und verwenden Sie A7 anstelle von SP.

Sobald Sie die Parameter eingegeben haben, übergibt MonAm die Kontrolle an den N-Befehl:

[N] Nächstes Vorkommnis suchen
Bei den Byte-, Wort- und Langwort-Suchbefehlen werden Sie immer eine Stelle finden: im MonAm-Speicher. Bei einem Text kann es Ihnen passieren, daß der immer noch im Tastaturpuffer steht. Die Suche können Sie mit der [Esc]-Taste abbrechen. Dazu fragt MonAm diese Taste alle 2 Byte (bei B, W und L) bzw. alle 64 Kbyte (bei Text) ab. Wenn eine Stelle gefunden wurde, werden die Anfangsadressen der Fenster aktualisiert. Lediglich Fenster 1 und Fenster 4 (bei Verwendung als Source-Fenster) bleiben unverändert.

Wenn Sie den [G]-Befehl im Source-Fenster verwenden, erscheint (bei erfolgreicher Suche) lediglich eine Meldung, in welcher Zeile der Text gefunden wurde. Falls keine Stelle gefunden wurde, verschwindet nur die Meldung »Suche ...«

[I] Intelligentes Kopieren
Mit diesem Befehl können Sie einen Speicherbereich in einen anderen kopieren. Geben Sie dazu bitte <Startadresse>, <Endadresse [inklusive]>, <Zieladresse> ein. Der Algorithmus ist insofern intelligent, als er den Speicher an eine Stelle kopieren kann, die sich mit dem Quellbereich überschneidet. Wenn Sie einen nichtexistierenden Bereich angeben, stürzt MonAm ab.

[L] Labels auflisten
In einem großen Fenster werden alle Labels aufgelistet. Mit einer beliebigen Taste blättern Sie weiter, mit [Esc] schließen Sie das Fenster wieder.

[Ctrl]+[U]<Name> Freigabe von Speicherplatz
Wenn Sie ein Programm mit Symboltabelle geladen haben und der Speicherplatz knapp wird, können Sie mit diesem Befehl den Speicherplatz z.T. wieder »freilegen«.

[W] Speicher füllen
Wenn Sie <Startadresse>, <Endadresse [inklusive]>, <Wert> angeben, wird der genannte Bereich mit <Wert> gefüllt werden.

[P] Disassemblieren
Mit diesem Befehl können Sie auf Diskette/zum Drucker disassemblieren. Dabei werden vorhandene Labels unterstützt; wenn Sie wollen, werden auch Cross-Reference-Labels unterstützt. Geben Sie dazu zunächst <Startadresse>,<Endadresse> ein. Die nächsten Angaben benötigt MonAm für einen Speicherbereich, in dem die Cross-Reference-Liste erstellt werden kann: <Pufferanfang>,<Pufferende>. Möchten Sie keine solche Liste, geben Sie einfach [Return] ein. Die nächste Zeile fragt nach einem Speicherbereich, der als DC-Direktiven disassembliert werden soll. Die Zeile sollte wie folgt aussehen: <Datenanfang>, <Datenende>. Optional können Sie dahinter (noch einmal durch ein Komma getrennt) die Größe der DCs angeben (also B, W oder L). Voreingestellt ist L. Schließlich werden Sie nach einem Dateinamen gefragt. Wenn Sie keinen eingeben, erfolgt die Ausgabe auf den Drucker.

Wird eine Cross-Reference-Liste erstellt, gibt es zunächst eine kurze Pause. Die Ausgabe kann jederzeit mit [Esc] abgebrochen werden.

[M] Adresse verändern
Dieser Befehl entspricht [A]+[A] und ist nur beibehalten worden, um mit der ersten Version des Debuggers kompatibel zu sein.

[O] Auswerten
Auch dieser Befehl stammt noch von der ersten Version des Debuggers und entspricht der Kombination [A]+[O].

[D] Laufwerk und Pfad ändern
Dieser Befehl ist wohl selbsterklärend.

Maschinensprache auf dem Amiga

Kapitel 1

Dieses erste Kapitel soll Ihnen, liebe Leser, eine Einführung in die Maschinensprache geben und anschließend noch auf einige Aspekte unseres DEVPAC-Assemblers eingehen. Zunächst möchte man fragen, gibt es eigentlich einen sachlichen Unterschied zwischen Maschinensprache und Assembler?

Nun, bei der reinen Maschinensprache handelt es sich um die »Muttersprache« des Prozessors unseres Computers. Sie ist extrem primitiv aufgebaut, denn sie besteht nur aus Nullen und Einsen. Im nächsten Abschnitt werden wir dies ausführlich erklären. Ein reines Maschinenprogramm stellt also nur eine Ansammlung von Nullen und Einsen dar. Um so ein umfangreiches Programm auf dem Amiga zu entwickeln, müßte man sich mit mehr als 500 000(!) solcher Zahlen herumquälen. Da selbst Albert Einstein wohl so den Spaß am Computer verloren hätte, führte man die Assemblersprache ein. Diese kann man mit keiner Hochsprache wie Basic vergleichen. Der einzige Unterschied zur Maschinensprache besteht darin, daß bestimmte Folgen von Nullen und Einsen, die für den Prozessor eine besondere Bedeutung haben, durch sogenannte

Mnemonics (griechisch: Die Kunst, das Gedächtnis durch Hilfen zu stärken)

dargestellt werden. Die Umwandlung von Assembler in Maschinensprache wird durch einen sogenannten Assembler (welch sinnvolle Bezeichnung...) durchgeführt. Hierbei handelt es sich um ein Programm, das »weiß«, welche Zahlenkombination welchem Mnemonic entspricht und dementsprechend diese Zahlen einsetzen kann. Die Trennung zwischen Maschinensprache und Assembler ist für uns Anwender damit bedeutungslos geworden, wir werden natürlich nur in Assembler programmieren. Um den Zusammenhang zu verdeutlichen, möchte ich Ihnen ein kleines Beispiel vorführen, dessen Wirkung Sie an dieser Stelle natürlich noch nicht verstehen müssen. Es soll ausschließlich zeigen, wie angenehm sich der Assembler im Vergleich zur reinen Maschinensprache ausmacht:

Maschinensprache	**Assembler**	**Wirkung auf den Prozessor**
00101100011110000001100101100101	move.l $1965,A6	Lädt Adreßregister 6 mit Inhalt des Speichers $1965

1.1 Der 68000-Mikroprozessor

Im Gegensatz zu höheren Programmiersprachen, in denen man gute Programme auch dann entwickeln kann, wenn man den Aufbau des Prozessors kaum oder gar nicht kennt, ist es für den Maschinenprogrammierer unerläßlich, sich mit dem »Gehirn« des Computers genauer zu beschäftigen. Der Mikroprozessor 68000 besitzt eine Reihe von sogenannten Registern, in denen alle Rechenoperationen ablaufen. Um den Aufbau eines solchen Registers zu verstehen, muß man wissen, daß der Prozessor mit elektrischen Strömen arbeitet und daher überhaupt nur zwei Zustände unterscheiden kann: Entweder es fließt Strom oder es fließt keiner. Als Programmierer hat man allerdings relativ wenig davon, zu wissen, ob Strom im Prozessor fließt. Deshalb rechnet man im sogenannten Dualsystem (Zahlensystem mit der Basis 2), das diese Zustände der Leitungen in die Mathematik übertragen kann, da auch hier genau zwei Zahlen existieren, nämlich die Null und die Eins. Zwischen Dualsystem und Prozessorströmen kann man daher folgende logische Verbindung herstellen:

Dualzahl	Spannung (V)	Strom fließt	Elektrischer Pegel
0	0-0.8	nein	Low
1	2.4-5	ja	High

Ein Prozessorregister besitzt mit einer Ausnahme 32 solcher »Schalter«, die man als Bits bezeichnet, da man mit einem Bit ja nur zwei verschiedene Werte charakterisieren könnte. Durch 32 Bit hingegen können 4 294 967 296 Zustände dargestellt werden. Dies ist nicht schwer zu verstehen, wenn man sich klarmacht, wie so ein Zahlensystem aufgebaut ist. Gewöhnlich rechnen wir im Dezimalsystem, dem Zahlensystem mit der Basis 10. Wenn wir z.B. die Zahl 156 betrachten, ist diese nur eine Abkürzung für den Rattenschwanz

$1 * 10^2 + 5 * 10^1 + 6 * 10^0$

Jede weiter nach links gerückte Stelle ergibt einen Wert, der um den Faktor 10 größer ist als der vorhergehende, d.h., der Exponent wird um eins erhöht. Genauso wird im Zweiersystem verfahren, der Unterschied besteht nur in der Basis des Zahlensystems und darin, daß die Faktoren nur die Zahlen Eins und Null annehmen können, während im Dezimalsystem Zahlen von Null bis Neun vorkommen können. So können wir die Binärzahl (wie die Zahlen des Dualsystems genannt werden) 111111 schreiben:

$1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 63$

Durch eine 32stellige Binärzahl können somit Zahlen von Null bis 4294967295 dargestellt werden, je nachdem, welche Bits gesetzt wurden. Auf dem Papier findet man oft eine solche Zahl mit Angaben der Wertigkeiten der einzelnen Bits vor, um schneller den Wert errechnen zu können:

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0	Bitposition
1 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1	Bit

Für die Wertigkeit der einzelnen Positionen gilt dabei:

Bitposition	Exponent	Wert	Bitposition	Exponent	Wert
0	2^{0}	1	16	2^{16}	65536
1	2^{1}	2	17	2^{17}	131072
2	2^{2}	4	18	2^{18}	262144
3	2^{3}	8	19	2^{19}	524288
4	2^{4}	16	20	2^{20}	1048576
5	2^{5}	32	21	2^{21}	2097152
6	2^{6}	64	22	2^{22}	4194304
7	2^{7}	128	23	2^{23}	8388608
8	2^{8}	256	24	2^{24}	16777216
9	2^{9}	512	25	2^{25}	33554432
10	2^{10}	1024	26	2^{26}	67108864
11	2^{11}	2048	27	2^{27}	134217728
12	2^{12}	4096	28	2^{28}	268435456
13	2^{13}	8192	29	2^{29}	536870912
14	2^{14}	16384	30	2^{30}	1073741824
15	2^{15}	32768	31	2^{31}	2147483648

Leider ist es für den Programmierer sehr umständlich, sich durch solche Zahlenwüsten von Nullen und Einsen kämpfen zu müssen. Deshalb wurde das sogenannte Hexadezimalsystem (Zahlensystem mit der Basis 16) eingeführt, das genau dieses Problem beseitigt. Die Idee besteht darin, jede 8-Bit-Binärzahl in zwei Teile, die sogenannten Nibbles, zu zerlegen. Jedes dieser Nibbles kann nun Werte von Null bis 15 annehmen, da die höchste Zweierpotenz jedes Nibbles 23 ist.

Dezimal	Binär	Hexadez.
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Dezimal	Binär	Hexadez.
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Jedem Nibble wird nun eine Hexadezimalzahl zugeordnet und diese beiden Zahlen aneinandergefügt. Da man nur Zahlen von null bis neun zur Verfügung hat, muß man für die Zahlen 10–15 eine andere Lösung finden. Diese besteht darin, einfach hierfür die Buchstaben A bis F zu benutzen.

Um zum Beispiel die Binärzahl 11110010 ins hexadezimale Format umzurechnen, geht man wie oben beschrieben vor: Nach der Aufteilung in die Nibbles 1111 und 0010 liest man die entsprechenden Hexadezimalzahlen F und 2 ab und setzt diese zusammen: F2.

Um eine 8-Bit-Binärzahl in diesem Format darzustellen, sind also nur zwei Stellen erforderlich. Um die einzelnen Zahlensysteme zu unterscheiden, stellt man einer Binärzahl ein %-Zeichen und der Hexadezimalzahl ein $-Zeichen vorweg, während die Dezimalzahl unverändert bleibt. Für unseren Prozessor sind drei Arten von Binärzahlen wichtig, weshalb man Abkürzungen für sie einführte:

8-Bit-Binärzahl	(1/4	Register)	: Byte
16-Bit-Binärzahl	(1/2	Register)	: Wort
32-Bit-Binärzahl	(1	Register)	: Langwort

Ein weiteres Problem für uns besteht darin, auch andere Zeichen verarbeiten zu können, wie z.B. Buchstaben. Unser Prozessor weiß natürlich überhaupt nicht, was ein Buchstabe ist, er selbst kann ja nur Stromzustände unterscheiden. Deshalb führte man den sogenannten ASCII-Code ein, in dem jedem Zeichen eine Zahl zwischen Null und 127 zugeordnet wird. ASCII ist die Abkürzung für American Standard Code for Information Interchange und benötigt 7 Bit. So wird dem Zeichen »A« der Code 65 zugewiesen, dem Zeichen »B« der Code 66, usw. Ein ASCII-Zeichen wird im Assemblerlisting durch das Einbinden in Hochkommas signalisiert.

Unser Prozessor enthält 18 der 32-Bit-Register sowie die schon erwähnte Ausnahme, ein 16-Bit-Register. Sehen wir uns den Aufbau und die Funktion der einzelnen Register an:

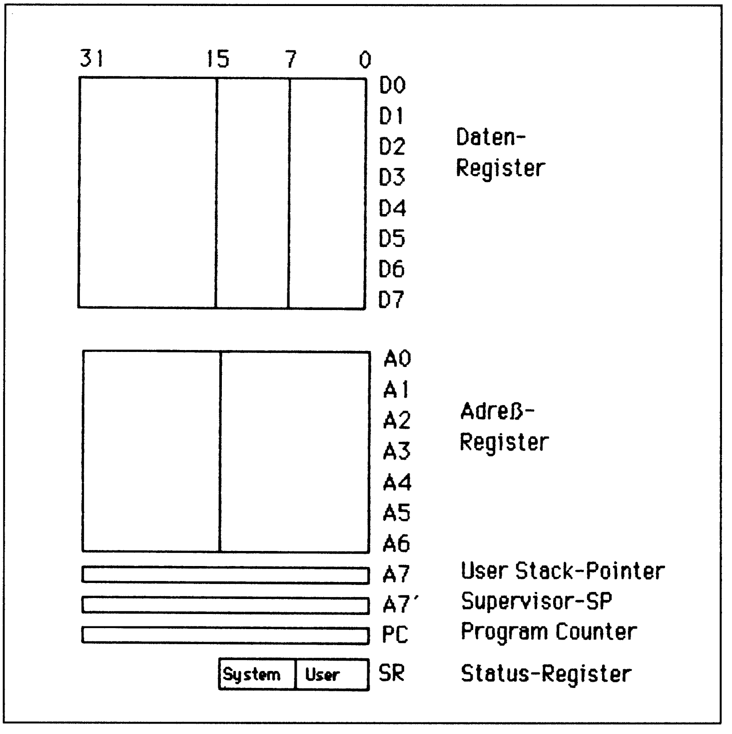


Bild 1.1: Das Registermodell des 68000

Die Datenregister D0–D7 können wahlweise mit 8-, 16- oder 32-Bit-Operationen arbeiten, was nichts anderes heißt, als man mit einem Maschinenbefehl entweder auf alle Bits des Registers oder aber auch nur auf die untersten 16 bzw. 8 Bit zugreifen kann. Dabei benötigen die Byte-Befehle grundsätzlich genausoviel Zeit zur Abarbeitung wie die Wort-Befehle, die ihrerseits aber schneller als die Langwort-Befehle ausgeführt werden können. Der Grund hierfür ist, daß der 68000 zwar intern mit 32 Bit rechnen kann, aber nur einen 16-Bit-Datenbus aufweist. Deshalb müssen die Langwörter aufgespalten und in zwei Teilen transportiert werden, während Bytes und Wörter in einem Stück übertragen werden können. Die Adreßregister A0–A6 können hingegen nur mit 16 oder 32 Bit adressiert werden. Ihre Bedeutung liegt darin, daß man mit Hilfe von bestimmten indirekten Adressierungsarten (siehe nächster Abschnitt) z.B. sehr einfach Tabellen abarbeiten kann. Prinzipiell kann man aber die Adreßregister auch als Datenregister verwenden, da sich arithmetische Operationen auch dort durchführen lassen. Hingegen ist es nicht möglich, ein Datenregister indirekt zu adressieren. Das Stackpoint-Register, auch Stapelzeiger genannt, enthält einen 32-Bit-Zeiger auf den sogenannten Stack, wobei zwischen dem User- und dem Supervisor-Modus unterschieden wird (siehe unten), die beide ihren eigenen Stapel benutzen. Beim Stack handelt es sich um einen speziellen Speicherbereich, der vom Prozessor als Speicher für wichtige Daten wie Rücksprungadressen von Unterprogrammen verwendet wird. Der Name Stapel kennzeichnet das Verfahren, mit dem dieser Speicher-

bereich verwaltet wird. Zunächst wird der Pointer um ein Langwort erniedrigt, damit nicht der Wert überschrieben wird, der ab der momentanen Adresse abgelegt ist. Danach wird der zu speichernde Wert an der neuen Adresse abgelegt. Dieses Verfahren wiederholt sich dann. Beim Lesen eines Wertes vom Stack wird umgedreht verfahren: Zunächst wird das Langwort aus der Adresse geholt, auf die der Stackpointer gerade zeigt. Daraufhin wird der Pointer um 4 (1 Langwort) erhöht, damit der folgende Wert gelesen werden kann. Aus diesem Verfahren ergibt sich, daß immer nur der zuletzt abgelegte Wert gelesen werden kann, genauso, wie man von einem Bücherstapel immer nur das oberste Buch wegnehmen kann (ohne daß der Stapel zusammenbricht...). Bild 1.1.2 erläutert das Stapelverfahren.

Das Statusregister wird in das User-Byte (untere 8 Bit) und das System-Byte (obere 8 Bit) aufgeteilt. Das User-Byte enthält 5 Bit, die als Flags bezeichnet werden. Sie werden vom Prozessor gesetzt und geben Auskunft über das Ergebnis des letzten ausgeführten Befehls, indem die Flags gesetzt oder gelöscht werden. Durch mehrere Maschinenbefehle kann man den Zustand der Flags einfach abfragen und z.B. einen Sprung auf Grund des Zustandes eines bestimmten Flags durchführen. Über die vielfältigen Möglichkeiten werden wir noch genau sprechen, deshalb soll hier nur kurz der Verwendungszweck angerissen werden:

Bit	**Name**	**Bedeutung**
0	Carry (C)	Übertragsbit. Es kann durch Verknüpfungen, Berechnungen und Verschiebungen verändert werden. Angezeigt werden Ober- und Unterläufe, d.h., das Ergebnis einer Operation läßt sich nicht mehr in dem gewählten Bereich (Byte, Wort, Langwort) darstellen.
1	Overflow (V)	Zeigt Divisionsoverflow oder eine Änderung des Vorzeichens bzw. einen Übertrag von Bit 6 in Bit 7 an. Hat geringere Bedeutung.
2	Zero (Z)	Nullbit. Zeigt an, daß das Ergebnis Null ist.
3	Negativ (N)	Vorzeichenbit. Es wird immer dann gesetzt, wenn das Ergebnis einer Operation negativ ist, d.h., wenn Bit 7 (Byte), 15 (Wort) oder 31 (Langwort) gesetzt sind.
4	Extended (X)	Wird nach arithmetischen Berechnungen eventuell gesetzt wie das Carry-Flag.
5-7	Unbenutzt	

Auf die Bits des System-Bytes kommen wir im nächsten Abschnitt zu sprechen. Als letztes Register haben wir noch den Programmzähler. Da unser Prozessor allein natürlich völlig hilflos ist, müssen wir ihm die Informationen, d.h. die Befehle und die zu verarbeitenden Daten, liefern. Dafür stellt der Amiga in der Grundversion 512 Kbyte (1 Kbyte = 1024 Byte) an Speicher zur Verfügung. Jede Speicherzelle ist 8 Bit »groß«, so daß ein einfacher Datenaustausch zwischen Speicher und Prozessorregistern möglich ist. Im Programmzähler befindet sich immer die Adresse, aus der der nächste Maschinenbefehl geholt werden soll. Jeder 68000-Befehl ist 16 Bit lang, so daß er ein Wort oder 2 Byte benötigt. Deshalb muß er auch unbedingt an einer geraden Adresse beginnen. Wenn Sie diese Tatsache ignorieren, wird es nicht lange dauern, bis sich der Guru für Ihre Tätigkeit zu interessieren beginnt. Es fällt auf, daß man theoretisch mit dem 32-Bit-Programmzähler nicht weniger als 4 Gigabyte an Speicher adressieren kann!

Schritt	SP vorher	Operation	SP nachher	Stackinhalt
1	Adresse	Keine	Adresse	Adresse :undefiniert
2	Adresse-4	Schieben	Adresse-4	Adresse :undefiniert Adresse-4: Wert 1
3	Adresse-8	Schieben	Adresse-8	Adresse :undefiniert Adresse-4: Wert 1 Adresse-8: Wert 2
4	Adresse-8	Holen	Adresse-4	Adresse-4: Wert 1 Adresse-8: undefiniert
5	Adresse-4	Holen	Adresse	Adresse: undefiniert Adresse-4: undefiniert

Bild 1.2: Nachführen des Stackpointers

1.1.1 Die Betriebszustände des 68000

Wie oben schon erwähnt enthält das Statusregister neben dem schon besprochenen User-Byte (Bits 0–7) noch das sogenannte System-Byte (Bits 8–15). In diesem Byte sind wiederum 5 Bit von Bedeutung, die man in drei Gruppen aufteilen kann:

- User-Supervisor-Bits
- Exception-Bits
- Interrupt-Bits

Im Gegensatz zum User-Byte kann der Eingriff in das System-Byte katastrophale Folgen haben, wenn man sich über die Funktion nicht genau im klaren ist, was man schon daran erkennen kann, daß die Exceptions für die beliebten Guru-Mediationen verantwortlich sind. Andererseits kann man durch korrekte Programmierung nicht nur den Guru beseitigen, wie der Monitor MONAM, sondern auch die Interrupts für seine Zwecke benutzen.

1.1.1.1 User- und Supervisor-Modus

Durch Bit 13 wird der Prozessor entweder im User (Bit gelöscht) oder im Supervisor-Modus (Bit gesetzt) betrieben. Der Unterschied besteht eigentlich nur darin, daß Sie im User-Modus, im dem sich der Prozessor normalerweise befindet, einige Befehle nicht benutzen dürfen. Hierbei handelt es sich um die privilegierten (für den Supervisior-Modus) Befehle. Nun könnte man denken, daß es ein leichtes sein müßte, das Modus-Bit einfach mit einem Maschinenbefehl zu setzten. Leider würde in diesem Fall aber der Guru zu einer Galavorstellung kommen, da es sich bei dem einzigen Befehl des 68000, mit dem man in das Statusregister hineinschreiben kann (siehe unten), um einen privilegierten Befehl handelt. Darum kann man wohl vom Supervisor- in den User-Modus umschalten, nicht aber umgedreht.

Deshalb integrierten die Entwickler des Amiga eine Routine in die Exec-Library, mit deren Hilfe man die Umschaltung durchführen kann. Da aber auch diese Herren nicht zaubern können, bedienten sie sich eines Tricks, den ich im nächsten Abschnitt erläutern werde.

Ein weiterer Unterschied zwischen beiden Modi liegt im verwendeten Stack-Bereich. Zwar wird auch weiterhin das Register A7 als Stapelzeiger verwendet, jedoch wird ein anderer Speicherbereich als Stapel benutzt. Beim Umschalten wird deshalb auch immer der Stackpointer mitverändert.

Des weiteren können auch Speicherzugriffe prinzipiell modusabhängig sein, so daß man als User eventuell auf gewisse Bereiche nicht zugreifen kann. Ist man darauf angewiesen, auf alle Speicherbereiche zuzugreifen und alle Befehle benutzen zu können, muß man auf jeden Fall den Supervisor-Modus benutzen.

1.1.1.2 Exceptions und Interrupts

Die Exceptions (Ausnahmen) können aus verschiedenen Gründen auftreten. Ein Grund liegt im Trace-Bit des Statusregisters (Bit 15). Ist dieses Bit gesetzt, wird nach jedem abgearbeiteten Befehl eine Exception ausgelöst, was für die Programmierer eines Maschinensprachemonitors natürlich sehr angenehm ist, da sie auf das sonst übliche und umständliche Verfahren verzichten können.

Bei einer Exception wird je nach Art des Ereignisses in eine bestimmte Routine verzweigt, nachdem folgende Schritte vorgenommen wurden:

- Retten des Statusregisters (SR) auf den Stack.
- Bit 13 im SR wird gesetzt (Supervisor-Modus) und Bit 15 gelöscht (kein Trace).
- Der Programmzähler und der User-Stackpointer werden gerettet.
- Das Super-Statuswort wird gerettet (siehe unten).
- Der Exception-Vektor, der auf die auszuführende Routine zeigt, wird geholt.
- Die Exception-Routine wird ausgeführt.

Die Vektoren liegen am Anfang des Speichers. Die folgende Tabelle zeigt den Zusammenhang zwischen der Adresse und der Exception-Ursache:

Exception-Nummer	Vektor-Adresse	Ursache
0	$0000	RESET: Anfangs-Stackpointer
1	$0004	RESET: Anfangs-Programmzähler
2	$0008	Bus-Error
3	$000C	Adreß-Error
4	$0010	Illegaler Befehl
5	$0014	Division durch Null
6	$0018	CHK-Befehl
7	$001C	TRAPV-Befehl
8	$0020	Privileg-Verletzung
9	$0024	Trace-Bit im Statusregister
10	$0028	Axxx-Befehlsemulation
11	$002C	Fxxx-Befehlsemulation
15	$003C	Uninitialisierter Interrupt
24	$0060	Unberechtigter Interrupt
25-31	$0064-$0083	Interrupt-Levels 1-7
32-47	$0080-$00BF	TRAP-Befehle
64-255	$0100-$03FF	User-Interrupt-Befehle

Wenn Sie Ihren Amiga einschalten, muß zunächst einmal ein Startzustand hergestellt werden, d.h., es muß irgendwo mit der Ausführung eines Programms begonnen werden. Die Adresse des Stackpointers und der Reset-Routine werden aus den ersten beiden Exception-Vektoren geholt. Es würde den Rahmen dieses Buches sprengen, genauer auf alle Exceptions einzugehen. Falls Sie sich für die programmtechnischen Möglichkeiten interessieren sollten, möchte ich Ihnen das »Referenz-Manual« empfehlen. In dem Super-Statuswort sind die Bits 0 bis 4 relevant:

Bit 0..2 Funktions-Code
Bit 3 0 = Gruppe 2, 1 = Gruppe 1
Bit 4 0 = Schreibzyklus wurde unterbrochen
1 = Lesezyklus wurde unterbrochen

Der Funktionscode gibt an, welchen Zugriff der Prozessor zum Zeitpunkt der Exception gerade vollzog. Dabei kann man grob zwischen einem Zugriff auf ein User- oder Supervisor-Programm sowie auf User- oder Supervisor-Daten unterscheiden, anders formuliert, zwischen einem Befehlswort und einem Datensatz. Die Einteilung in Gruppen ist deshalb von Bedeutung, weil theoretisch in der Abarbeitung einer Exception-Routine eine weitere Unterbrechung möglich wäre, z.B. wenn Sie innerhalb dieser Routine durch Null dividieren. Daher wurden für die einzelnen Exceptions Prioritäten gesetzt.

Gruppe 0	Gruppe 1	Gruppe 2
Reset Bus-Error Adreß-Error	Trace Interrupt Illegaler Befehl Trap $Axxx, Trap $Fxxx Privilegverletzung	Trap #n Trap v CHK Division durch Null

Eine Exception kann nur durch eine weitere höhere Priorität unterbrochen werden, z.B. ein illegaler Befehl durch einen Bus-Error. Die Einteilung ist sehr sinnvoll, da ein Maschinenprogramm nach den Gruppe 0-Exceptions überhaupt nicht mehr korrekt weiterarbeiten kann, weshalb eine Divison durch Null auch nicht mehr interessiert.

Zwei Antworten bin ich Ihnen noch schuldig geblieben: Wie schafft es Exec, in den Supervisor-Modus zu schalten, und was ist beim Amiga die Folge einer Exception? Zunächst zu Exec. Der schon erwähnte Trick besteht einfach darin, ohne Rücksicht auf Verluste das Statusregister im User-Modus zu beschreiben. Durch die ausgelöste Exception (Privilegverletzung) wird automatisch in den Supervisor-Modus umgeschaltet, womit das eigentliche Ziel schon erreicht wäre. Es stört jedoch die Exception-Routine. Hier wird nun geprüft, wodurch die Exception ausgelöst wurde. Stellt sich heraus, daß die Exec-Routine die Ursache war, wird einfach in das unterbrochene Programm zurückverzweigt. Die Routine mit dem Namen Superstate wird einfach mit dem Makroaufruf

```
CALLEXEC SuperState
move.l d0,stackpointer
stackpointer  dc.l 0
```

gestartet. Zurückgegeben wird hierbei der letzte Wert des Stackpointers, den wir dann benötigen, wenn wir wieder in den User-Modus zurückschalten möchten. Dies könnten wir durch Löschen des Bits 13 im Statusregister realisieren, da wir im Supervisor-Modus den privilegierten Befehl benutzen dürfen. Einfacher ist aber der Aufruf einer weiteren Exec-Routine:

```
move.l stackpointer,d0
CALLEXEC UserState
```

Übergeben wird hier der alte Wert des Stackpointers, den uns die SuperState-Funktion geliefert hat. Nun aber zu den Folgen einer Exception: Es wird Sie wahrscheinlich nicht überraschen, daß am Ende aller Unterbrechungsroutinen eine Guru-Meditation steht. Vorher erscheint ein Requester, in dem man den weiteren Verlauf der Aktion festlegen kann. Hierzu muß man sagen, daß die scheinbare Auswahl zwischen der Kapitulation durch das Anklicken von »Cancel« und dem Versuch, noch etwas zu retten (»Retry«) nur als Makulatur zu bezeichnen ist. In Wirklichkeit macht das Betriebssytem nämlich keinen ernsthaften Versuch, einen Absturz zu vermeiden. Dabei ist der Rechner noch keineswegs am Ende, wenn der Requester erscheint, sondern erst, wenn sich das Guru-Alert meldet, wie

man durch einen einfachen Trick zeigen kann: Falls Sie überhaupt nicht auf den Requester reagieren, sondern in einen anderen Task umschalten, können Sie dort seelenruhig weiterarbeiten, um z.B. wichtige Daten auf die Diskette zu schreiben. Daß man aber auch in dem (angeblich) verlorenen Task weiterarbeiten kann, zeigt uns der Monitor MonAmiga: Er benutzt eigene Exception-Routinen, indem er die Vektoren auf diese umgeleitet hat. Sie fangen sämtliche Exceptions (auch die der Gruppe 0!) ab. Man kann dieses Verfahren natürlich auch in eigenen Programmen verwenden, wozu aber gute Betriebssystemkenntnisse erforderlich sind. Eine eigene Exception-Routine muß immer mit einem RTE-Befehl abgeschlossen werden.

Ähnlich wie eine Exception verhält sich der Ablauf bei einem Interrupt: Auch werden Statusregister etc. auf den Stack geschoben, bevor in die Unterbrechungsroutine gesprungen wird, die ebenso mit einem RTE-Befehl abgeschlossen werden muß. Die Ursache liegt allerdings auf ganz anderer Ebene, den Peripherie-Bausteinen. Sie haben Zugriff auf spezielle Pins des 68000-Prozessors, die hardwaremäßig folgende Aktionen auslösen: Zunächst wird festgestellt, welcher Interrupt ausgelöst wurde (0–7). Diese Zahl wird mit dem sogenannten Interrupt-Level verglichen, einer Zahl, die sich durch die Bits 8–10 des Statusregisters zusammensetzt und ebenfalls die Werte 0 bis 7 annehmen kann. Ist der ausgelöste Interrupt höherwertiger als der Interrupt-Level, wird in die entsprechende Routine verzweigt, sonst wird er ignoriert. Der Grund für dieses Verfahren liegt in der Wichtigkeit der möglichen Interrupt-Auslöser: So ist es für den Amiga lebenswichtig, in konstanten Abständen seine Speicherbausteine zu refreshen, um einen Absturz zu vermeiden. Wenn in dieser Zeit der Anwender einen Tastatur-Interrupt auslöst, indem er eine Taste betätigt, muß er natürlich ignoriert werden, weshalb der Tastatur-Interrupt auch eine niedrigere Priorität besitzt.

1.2 Die Adressierungsarten des 68000

Wie wir im letzten Abschnitt festgestellt haben, wird jeder Prozessorbefehl durch 16 Bit dargestellt, so daß 65 536 verschiedene Kombinationen möglich wären. Leider existieren aber nur 100 verschiedene Befehle, die zwar theoretisch ausreichen würden, um ein Maschinenprogramm zu entwerfen, wahrscheinlich aber auch dazu führen würden, daß der Programmierer anschließend in eine Irrenanstalt eingewiesen würde. Erst durch die verschiedenen Adressierungsarten der einzelnen Befehle ergeben sich fast 1000 verschiedene Kombinationen, mit denen man sehr gut leben kann. Ich möchte Ihnen nun die zwölf Adressierungsarten des 68000 an Hand des Move-Befehls verdeutlichen. Dieser am häufigsten benötigte Befehl kopiert Daten von einer Adresse zu einer anderen. Der Name »Move« ist daher etwas irreführend, da man unter dem Verschieben von Daten im allgemeinen versteht, daß nach der Operation der Quellbereich gelöscht ist.

1.2.1 Die unmittelbare Adressierung

Beispiel: `move #$10,d0`

Das Register D0 wird mit dem Wert geladen, den die Speicherzellen enthalten. Die »$10« ist also selbst Bestandteil des Programms. Wenn der Prozessor nun auf einen Befehlscode mit unmittelbarer Adressierung trifft, interpretiert er automatisch den Inhalt der folgenden Speicherzellen als den zu ladenen Wert. Deshalb wird durch Erhöhung des Programmzählers dafür gesorgt, daß die Speicherzellen übersprungen und nicht irrtümlich als nächster Maschinenbefehl angesehen werden.

1.2.2 Register-Direkt-Adressierung

Beispiel: `move d2,d0`

Der Inhalt eines Registers wird in ein anderes Register kopiert. Je nachdem, ob es sich um eine Byte-, Wort,- oder Langwort-Adressierung handelt, werden die untersten 8 bzw. 16 Bit oder alle 32 Bit kopiert, die nicht angesprochenen Bits bleiben unbeeinflußt.

1.2.3 Adreßregister-Indirekt-Adressierung (ARI)

Beispiel: `move (a0),d0`

Hier wird nicht der Inhalt des Registers a0, sondern der Adresse, auf die das Register a0 zeigt, in das Datenregister d0 kopiert. Abhängig davon, ob es sich um eine Wort- oder Langwort-Adressierung handelt, werden die ersten beiden oder 4 Byte ab der jeweiligen Adresse genommen, wenn z.B. das Register a0 den Wert $00C01234 beinhalten würde und ab der Adresse $00C01234 die Bytes $FF,$23,$19,$65 abgelegt wären, würde der Befehl move.w (a0),d0 nur die Bytes $19 und $65 kopieren, während durch die Langwort-Adressierung move.l (a0),d0 die Bytes $FF,$23,$19 und $65 in das Register d0 kopiert würden.

1.2.4 ARI-Adressierung mit Postinkrement

Beispiel: `move (a0)+,d0`

Hierbei wird nach der Ausführung des Datenkopierers der Inhalt des Adreßregisters je nach Adressierungsart um 1 (Byte), 2 (Wort) oder 4 (Langwort) Byte erhöht, so daß in unserem obigen Beispiel das Adreßregister die Werte $00C01235 (Byte), $00C01234 (Wort) oder $00C01238 (Langwort) enthalten würde.

1.2.5 ARI-Adressierung mit Predekrement

Beispiel: `move -(a0),d0`

Ähnlich wie die letzte Adressierungsart wird hier aber vor dem Kopiervorgang der Inhalt des Adreßregisters um 1 (Byte), 2 (Wort) oder 4 (Langwort) Byte erniedrigt.

1.2.6 ARI-Adressierung mit Adreßdistanz

Beispiel: `move 100(a0),d0`

Der Inhalt der Adresse, die sich aus der Summe des Inhalts des Registers a0 und dem Wert 100 ergibt, wird in das Register d0 kopiert, so würde mit dem Adreßregisterinhalt $00C01234 die Adresse $00C01298 angesprochen. Die Adreßdistanz stellt eine vorzeichenbehaftete 16-Bit-Zahl im Bereich von -32768 bis +32767 dar.

1.2.7 ARI-Adressierung mit Adreßdistanz und Index

Beispiel: `move -100(a0,d1),d0`

Hierbei wird die Adresse, aus der schließlich das Wort oder Langwort in das Register d0 kopiert wird, aus drei Komponenten ermittelt, die allesamt addiert werden: Zunächst haben wir den Inhalt des Adreßregisters a0, zu dem der Inhalt des Datenregisters d0 addiert wird. Bei letzterem kann man zwischen einer Byte,- Wort- oder Langwort-Adressierung wählen, während beim Adreßregister natürlich nur Wort und Langwort zulässig sind. Zu der Summe dieser beiden Registerinhalte wird nun wiederum eine Konstante addiert, die aber nur noch eine vorzeichenbehaftete 8-Bit-Zahl von -128 bis +127 einnehmen kann. Die endgültige Summe stellt nun die Adresse dar, auf die zugegriffen wird. Diese Adressierungsart ist sehr nützlich zur Tabellenabarbeitung und kann auf anderen Prozessoren wie z.B. dem 6510 des C64 nur sehr schwer simuliert werden.

1.2.8 Absolut-Kurz- und -Lang-Adressierung

Beispiel: `move $FF000000,$EE000000`

Der Inhalt des Speichers ab der Adresse $FF000000 wird in den Bereich ab $EE000000 kopiert. Bei der kurzen Adressierung sind dies die ersten beiden Bytes (Die Inhalte von $FF000000 und $FF000001, die nach $EE000000 und $EE000001 kopiert werden), bei der langen wird das erste Langwort kopiert (die Inhalte von $FF000000-$FF000003 nach $EE000000-$EE000003).

1.2.9 PC-Relative Adressierung mit Adreßdistanz

Beispiel: `move 100(PC),d0`

Diese Adressierungsart funktioniert ähnlich wie die ARI-Adressierung mit Adreßdistanz, allerdings wird hier keine feste Adresse aus einem Register geholt, sondern mit dem aktuellen Stand des Programmzählers gearbeitet. Da beim Amiga die Programme keinen festen Platz haben, sondern jedesmal an eine andere Stelle geladen werden können, dürfte man normalerweise überhaupt keine festen Adressen verwenden, sondern müßte sein Programm nur mit PC-relativen Befehlen aufbauen. Als Anwender des DEVPAC-Assemblers interessiert uns dies aber herzlich wenig. Er generiert nämlich zu jedem Programm eine Tabelle mit allen verwendeten absoluten Adressen, die nach dem Laden des Programms an den jeweils verwendeten Bereich angepaßt werden. Ein mit DEVPAC geschriebenes Programm läuft daher immer überall im Speicher (man sagt, es ist »relokatibel«). Für die bedauernswerten Menschen, die trotzdem PC-relativ programmieren möchten, sei noch gesagt, daß auch hier der Offset durch eine vorzeichenbehaftete 16-Bit-Zahl dargestellt wird.

1.2.10 PC-Relative Adressierung mit Adreßdistanz und Index

Beispiel: `move 100(PC,a0),d0`

Hierfür gilt das für Punkt 1.2.8 gesagte, die Basisadresse wird aber durch den Inhalt des Programmzählers +2 ermittelt. Ebenso wie auf die letzte Adressierungsart können wir als DEVPAC-Anwender hierauf verzichten, so daß Sie in diesem Buch keinen einzigen Befehl mit PC-Adressierung vorfinden werden.

1.3 Die 68000-Befehle

1.3.1 Die Codierung eines 68000-Befehls

Wenn man ein sich im Speicher befindliches Maschinenprogramm ansehen möchte, benutzt man im allgemeinen die Disassembler-Funktion des MONAMIGA-Monitors. Falls man sich jedoch nur die einzelnen Byte-Inhalte in hexadezimaler Form ausgeben läßt, wird man, anders als beim 6510, das Maschinenprogramm kaum analysieren können, da es zu viele verschiedene Befehlskombinationen gibt. Dabei ist das 16-Bit-Befehlswort durchaus nicht zufällig gewählt, sondern nach einem logischen Verfahren aufgebaut: Der eigentliche Befehl (z.B. Move) wird durch die vier höchstwertigen Bits dargestellt, d.h., daß diese Bits unabhängig von der gewählten Adressierung sind. Die übrigen 12 Bit teilen sich in zwei Gruppen mit jeweils 6 Bit auf, wobei die Bits 6–11 die Adressierungsart der Quelle und die Bits 0–5 diejenige des Ziels (falls vorhanden) repräsentieren.

Diese 6 Bit sind jeweils nochmal in zwei Gruppen mit 3 Bit aufgeteilt, wobei die Gruppen mit den höherwertigen 3 Bit »Modus« genannt wird, während die Gruppe mit den nieder-

wertigen Bits »Register« heißt. Die folgende Tabelle zeigt den Zusammenhang zwischen den Bits und der Adressierungart sowie dem benutzten Register:

Adressierungsart	**Kürzel**	**Modus-Bit**	**Register-Bit**	
Datenregister direkt	Dn	000	Dn	0<=Dn<=7
Adreßregister direkt	An	001	An	0<=An<=7
Adreßregister indirekt (ARI)	(An)	010	An	0<=An<=7
ARI mit Postinkrement	(An)+	011	An	0<=An<=7
ARI mit Predekrement	-(An)	100	An	0<=An<=7
ARI mit Adreßdistanz	d16(An)	101	An	0<=An<=7
ARI mit Adreßdistanz+Index	d8(An,Rn)	110	An	0<=An<=7
Absolut kurz	$XXXX	111	000	
Absolut lang	$XXXXXXXX	111	001	
PC-Relativ mit Adreßdistanz	d16(PC)	111	010	
PC-Rel. m. Adr.-Distanz+Index	d8(PC,Rn)	111	011	
Unmittelbar	#	111	100	

1.3.2 Übersicht und Funktion aller 68000-Befehle

In diesem Abschnitt werde ich Ihnen nun alle Befehle des 68000 vorstellen. Neben der Erklärung wird jeder Befehl in einer Tabelle der folgenden Form aufgelistet, aus der alle wichtigen Informationen sofort abzulesen sind:

Befehl :	Funktion:
ea Dn An (An) (An)+ -(An) d(An) d(An,Rn) $.W $.L d(PC) d(PC,Rn) # B W L	
Flags X N Z V C	Bemerkungen:

Befehl
mnemonische Form des Befehls

Funktion
Beschreibung der Wirkung des Befehls
Dabei bedeuten:

ea	Effektive Adresse
Dn	Datenregister D1 bis D7 (1 <= n <= 7)
An	Adreßregister A1 bis A7 (1 <= n <= 7)
Rn	Daten- oder Adreßregister 1 bis 7 (1 <= n <= 7)
d	Adreßdistanz
xK	Konstante mit maximal (x=###:32, x=##:16, x=#:8) Bits
PC	Programmzähler
SR	Statusregister
CCR	Condition-Code-Register (Flags)
SSP	Supervisor-Stackpointer
USP	User-Stackpointer (Register A7)
B	Anzahl Taktzyklen der Adressierungsart bei Byte-Adressierung
W	Anzahl Taktzyklen der Adressierungsart bei Wort-Adressierung
L	Anzahl Taktzyklen der Adressierungsart bei Langwort-Adressierung
*	Adressierungsart kann nicht benutzt werden
X	Extended-Flag
N	Negativ-Flag
Z	Zero-Flag
V	Overflow-Flag
C	Carry-Flag

Flags
Befindet sich unter der Abkürzung des Flags ein Eintrag, so wird dies durch den Befehl beeinflußt. Dabei bedeutet:

+	Flag wird vom Ergebnis der Operation beeinflußt.
1	Flag wird gesetzt.
0	Flag wird gelöscht.

Bemerkungen
Spezielle Anmerkungen zu den Befehlen.

1.3.2.1 Die Transferbefehle

Seiner Bedeutung angemessen steht der MOVE-Befehl in verschiedenen Variationen zur Verfügung. Besonders interessant sind der MOVEM-Befehl, mit dem die Inhalte mehrerer Register gleichzeitig verschoben werden können, und der MOVEP-Befehl, mit dem man externe Peripheriebausteine ansprechen kann. Bei letzterem werden die Daten byteweise übertragen, um z.B. Zähler im Format High-Low laden zu können. Ebenfalls sehr nützlich ist der EXG-Befehl, mit dessen Hilfe man die Inhalte von zwei Registern austauschen kann, ohne eine Zwischenspeicherung vorzunehmen. So kann man die Befehlssequenz

```
MOVE D1,D2
MOVE D3,D1
MOVE D2,D3
```

einfach durch den Befehl

```
EXG D1,D3
```

ersetzen, das Register D2 braucht nicht als Zwischenspeicher benutzt zu werden. Der SWAP-Befehl wird dazu benutzt, innerhalb eines Registers die Bits 0 bis 15 mit den Bits 16 bis 31 auszutauschen. Die Befehle LINK und UNLINK dienen dazu, einen lokalen Stack auf- und abzubauen. Als »normaler« Maschinenprogrammierer kann man aber gut darauf verzichten, für Hochsprachen-Compiler sind sie jedoch bedeutsam, wenn mit lokalen Variablen (innerhalb eines Unterprogramms) gearbeitet wird. Während man z.B. den Befehl LINK A2,#50 durch die Zeilen

```
move.l A2,-(SP)
move.l SP,A2
add.l #50,SP
```

ersetzen könnte, wodurch ein 50 Byte großer Bereich als lokaler Stack definiert würde, könnte man den Befehl UNLK A2, der unbedingt am Ende des Unterprogramms aufgerufen werden müßte, durch die Befehle

```
move.l A2,SP
move.l (SP)+,A2
```

simulieren. Damit hätten sowohl das Register A2 als auch der Stackpointer wieder den alten Inhalt vor dem LINK-Befehl. Verheerend wäre es, den UNLK-Befehl wegzulassen, da das Hauptprogramm dann mit einem falschen Stapel operieren würde. Es wäre mehr als seltsam, wenn daraus keine Guru-Meditation entstände!

Befehl: MOVE ea,(An)												Funktion: Kopiere Daten nach (An)
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	12	12	14	16	18	16	20	16	18	12
W	8	8	12	12	14	16	18	16	20	16	18	12
L	12	12	20	20	22	24	26	24	28	24	26	20
Flags	X	N	Z	V	C	Bemerkungen: Keine						
		+	+	0	0							

Befehl: MOVE ea,(An)+												Funktion: Kopiere Daten nach (An)+
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	12	12	14	16	18	16	20	16	18	12
W	8	8	12	12	14	16	18	16	20	16	18	12
L	12	12	20	20	22	24	26	24	28	24	26	20
Flags	X	N	Z	V	C	Bemerkungen: Keine						
		+	+	0	0							

Befehl: MOVE ea,-(An)												Funktion: Kopiere Daten nach -(An)
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	12	12	14	16	18	16	20	16	18	12
W	8	8	12	12	14	16	18	16	20	16	18	12
L	14	14	20	20	22	24	26	24	28	24	26	20
Flags	X	N	Z	V	C	Bemerkungen: Keine						
		+	+	0	0							

Bild 1.3: Die Transferbefehle MOVE ea,(An); MOVE ea,(An)+ ; MOVE ea,-(An)

Befehl: MOVE ea,d(An)	Funktion: Kopiere Daten nach d(An)

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	12	*	16	16	18	20	22	20	24	20	22	16
W	12	12	16	16	18	20	22	20	24	20	22	16
L	16	16	24	24	26	28	30	28	32	28	30	24

Flags	X	N	Z	V	C	Bemerkungen: $-32768 < d < +32767$
		+	+	0	0	

Befehl: MOVE ea,d(An,Rn)	Funktion: Kopiere Daten nach d(An,Rn)

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	14	*	18	18	20	22	24	22	26	22	24	18
W	14	14	18	18	20	22	24	22	26	22	24	18
L	18	18	26	26	28	30	32	30	34	30	32	26

Flags	X	N	Z	V	C	Bemerkungen: $-128 < d < +127$
		+	+	0	0	

Befehl: MOVE ea,Dn	Funktion: Kopiere Daten in Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	12	14	8
W	4	4	8	8	10	12	14	12	16	12	14	8
L	4	4	12	12	14	14	18	16	20	16	18	12

Flags	X	N	Z	V	C	Bemerkungen: Keine
		+	+	0	0	

Bild 1.4: Die Transferbefehle MOVE ea,d(An); MOVE ea,d(An,Rn); MOVE ea,Dn

Befehl: MOVE ea,$.W | Funktion: Kopiere Daten nach Wortadresse

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	12	*	16	16	18	20	22	20	24	20	22	16
W	12	12	16	16	18	20	22	20	24	20	22	16
L	16	16	24	24	26	28	30	28	32	28	30	24

Flags	X	N	Z	V	C
		+	+	0	0

Bemerkungen: Keine

Befehl: MOVE ea,$.L | Funktion: Kopiere Daten nach Langwortadresse

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	16	*	20	20	22	24	26	24	28	24	26	20
W	16	16	20	20	22	24	26	24	28	24	26	20
L	20	20	28	28	30	32	34	32	36	32	34	28

Flags	X	N	Z	V	C
		+	+	0	0

Bemerkungen: Keine

Befehl: MOVE ea,CCR | Funktion: Kopiere Daten in Condition-Code-Register

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	12	*	16	16	18	20	22	20	24	20	22	16
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Einziger Befehl für direkten Zugriff auf Flags im User-Modus.

Bild 1.5: Die Transferbefehle MOVE ea,$.W; MOVE ea,$.L; MOVE ea,CCR

Befehl: MOVE ea,SR	Funktion: Kopiere Daten in Statusregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	12	*	16	16	18	20	22	20	24	20	22	16
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Privilegierter Befehl, darf nur im Supervisor-Modus benutzt werden.
	+	+	+	+	+	

Befehl: MOVE SR,ea	Funktion: Kopiere Statusregister nach ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	6	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Keine

Befehl: MOVEA ea,An	Funktion: Kopiere Daten in Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	4	4	8	8	10	12	14	12	16	12	14	8
L	4	4	12	12	14	16	18	16	20	16	18	12

Flags	X	N	Z	V	C	Bemerkungen: Keine

Bild 1.6: Die Transferbefehle MOVE ea,SR; MOVE SR,ea; MOVEA ea,An

Befehl: MOVE USP,An	Funktion: Kopiere Userstackpointer in Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	4	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Privilegierter Befehl, darf nur im Supervisor-Modus benutzt werden.

Befehl: MOVE An,USP	Funktion: Kopiere Adreßregister in Userstackpointer

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	4	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Privilegierter Befehl, darf nur im Supervisor-Modus benutzt werden.

Befehl: MOVEM ea,RL	Funktion: Kopiere Daten ab ea in Registerliste

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	*	16	18	16	20	16	18	*
L	*	*	12	12	*	16	18	16	20	16	18	*

Flags X N Z V C	Bemerkungen: Beispiel Aufbau Registerliste: D1-D5/A2-A6 = Register D1 bis D5 sowie A2 bis A6

Bild 1.7: Die Transferbefehle MOVE An,USP; MOVE USP,An; MOVEM ea,RL

Befehl: MOVEM RL,ea	Funktion: Kopiere Daten einer Registerliste ab ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	8	*	8	12	14	12	16	*	*	*
L	*	*	8	*	8	12	14	12	16	*	*	*

Flags X N Z V C	Bemerkungen: siehe MOVEM ea,RL

Befehl: MOVEP Dn,ea	Funktion: Kopiere Daten in 8-Bit-Peripheriebaustein

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	16	*	*	*	*	*	*
L	*	*	*	*	*	24	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Die Daten werden byteweise übertragen, begonnen wird mit dem höchstwertigen Byte

Befehl: MOVEP ea,Dn	Funktion: Hole Daten von 8-Bit-Peripheriebaustein

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	16	*	*	*	*	*	*
L	*	*	*	*	*	24	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: siehe MOVEP Dn,ea

Bild 1.8: Die Transferbefehle MOVEM RL,ea; MOVEP Dn,ea; MOVEP ea,Dn

Befehl: MOVEQ ##K,Dn	Funktion: Kopieren einer Konstanten in Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	4
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Vorzeichenerweiterung auf 32 Bit
		+	+	0	0	

Befehl: LEA ea,An	Funktion: Laden von ea in ein Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	4	*	*	8	12	8	12	8	12	*

Flags	X	N	Z	V	C	Bemerkungen: Im Gegensatz zum MOVE-Befehl wird eine Adresse, nicht deren Inhalt geladen.

Befehl: PEA ea	Funktion: Ablegen von ea auf dem Stapel <-(SP)>

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	12	*	*	16	20	16	20	16	20	*

Flags	X	N	Z	V	C	Bemerkungen: Entspricht PHA-Befehl beim 6510

Bild 1.9: Die Transferbefehle MOVEQ ##K,Dn; LEA ea, An; PEA ea

Befehl: EXG Dn,Dn	Funktion: Tauschen des Inhalts zweier Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	6	*	*	*	*	*	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Keine

Befehl: EXG An,An	Funktion: Tauschen des Inhalts zweier Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	6	*	*	*	*	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Keine

Befehl: EXG An,Dn	Funktion: Tausch zwischen Daten- und Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	6	*	*	*	*	*	*	*	*	*	*	*

Flags X N Z V C	Bemerkungen: Keine

Bild 1.10: Die Transferbefehle EXG Dn,Dn; EXG An,An; EXG An,Dn

Befehl: SWAP Dn	Funktion: Tausch der Bits 0-15 mit den Bits 16-31

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	4	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Keine
		+	+	0	0	

Befehl: LINK An,#bytes	Funktion: Anlegen eines neuen Stapelbereiches

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Funktion: 1.) An -(SP) 2.) SP An 3.) SP+#bytes SP. #Bytes legt die Stackgröße fest.

Befehl: UNLK An	Funktion: Auflösen eines mit LINK erzeugten Stapels

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	12	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Funktion: 1.) An SP 2.) (SP)+ An An und SP haben die Inhalte vor dem LINK-Befehl.

Bild 1.11: Die Transferbefehle SWAP Dn; LINK An,#bytes; UNLK An

1.3.2.2 Die arithmetischen Befehle

Der Sinn eines Computers besteht darin, Rechenoperationen in extrem kurzer Zeit ausführen zu können. Deshalb wäre er ohne die Rechenbefehle ein nutzloser Haufen von Elektronik. Nun darf man allerdings nicht annehmen, daß der Prozessor beliebige Rechnungen wie z.B. das Ziehen einer Quadratwurzel durchführen kann. Die Umsteiger vom C64 werden sich erinnern, daß der 6510-Prozessor nur zwei Befehle zur Addition und Subtraktion von jeweils 8-Bit-Zahlen aufweist. Im Gegensatz dazu scheint der 68000 ein Rechengenie zu sein. Er kann Addition und Subtraktion nicht nur mit 32-Bit-Zahlen durchführen, was einem die vom 6510 bekannten langwierigen Übertragsrechnungen in fast allen Fällen ersparen wird, sondern sogar zwei 16-Bit-Zahlen miteinander multiplizieren oder eine 32-Bit-Zahl durch eine 16-Bit-Zahl dividieren! Alle weiteren Rechenoperationen müssen aber auch bei diesem Prozessor durch teilweise komplizierte Maschinenprogramme erfolgen. Die Addition erfolgt bitweise, wobei man zwischen drei Fällen unterscheiden muß:

0+0 = 0
0+1 = 1
1+1 = 0 plus Übertrag

Wie bei der dezimalen Addition werden also die einzelnen Stellen verknüpft, wobei ein Übertrag in der folgenden Stelle berücksichtigt wird. Durch den riesigen Wertebereich von 32 Bit wird man praktisch nie mit dem Problem konfrontiert werden, daß ein Übertrag in der 32. Stelle auftritt. Dieses 33. Bit kann nicht mehr in dem 32-Bit-Register untergebracht werden und wird deshalb in das Extended-Flag hineingeschrieben. Um das X-Flag auch in einer weiteren Rechnung berücksichtigen zu können, existiert neben der normalen Form des ADD-Befehls noch eine weitere Version (ADDX), die zu den beiden Summanden noch das X-Flag hinzuaddiert. Damit kann man beliebig große Zahlen addieren, soweit man sie in 32-Bit-Teile zerlegt hat. Man beginnt dabei immer mit den niederwertigsten 32 Bit. Das folgende Beispiel demonstriert die Addition der Zahlen $1240000000 und $1FF000000:

1. Schritt: Zerlegung in 32-Bit-Teile

```
move.l #$00000012,d0    ;1.Zahl, höherwertige 32 Bit
move.l #$40000000,d1    ;1.Zahl, niederwertige 32 Bit
move.l #$00000001,d2    ;2.Zahl, höherwertige 32 Bit
move.l #$FF000000,d3    ;2.Zahl, niederwertige 32 Bit
```

2. Schritt : Addition der niederwertigen 32 Bit

```
add.l d1,d3             ;d1 + d3 nach d3 schreiben
                        (X-Flag wegen Übertrags auf 1)
```

3. Schritt : Addition der höherwertigen 32 Bit

```
addx.l d0,d2            ;d0 + d2 + x-Flag nach d2 schreiben
```

Damit steht in den Registern d2/d3 die 64-Bit-Zahl $143F000000 zur Verfügung. Analog zur Addition wird die Subtraktion durchgeführt, hier wird bitweise subtrahiert. Man muß vier Fälle unterscheiden:

0–0 = 0
1–0 = 1
0–1 = 1 plus Unterlauf
1–1 = 0

Die Unterläufe werden automatisch berücksichtigt, solange sie sich im 32-Bit-Bereich bewegen. Ein Unterlauf des 0. Bit wird in das X-Flag übertragen. Deshalb existiert auch hier eine Version des SUB-Befehls, bei der neben der eigentlichen Zahl zusätzlich das X-Flag abgezogen wird. Wie bei der Addition müssen zunächst die unteren 32 Bit behandelt werden. Unser obiges Beispiel, übertragen auf die Subtraktion, müßte daher wie folgt programmiert werden:

1. Schritt: Zerlegung in 32-Bit-Teile

```
move.l #$00000012,d0   ;1.Zahl, höherwertige 32 Bit
move.l #$40000000,d1   ;1.Zahl, niederwertige 32 Bit
move.l #$00000001,d2   ;2.Zahl, höherwertige 32 Bit
move.l #$FF000000,d3   ;2.Zahl, niederwertige 32 Bit
```

2. Schritt : Subtraktion der niederwertigen 32 Bit

```
sub.l d3,d1            ;d1 - d3 nach d1 schreiben
                       (X-Flag wegen Unterlauf auf 1)
```

3. Schritt : Subtraktion der höherwertigen 32 Bit

```
subx.l d2,d0           ;d0 - d2 - x-Flag nach d0 schreiben
```

Damit steht in den Registern d0/d1 die 64-Bit-Zahl $1041000000 zur Verfügung. Neben den Befehlen für die bitweise Addition/Subtraktion stehen noch weitere für die Bearbeitung von sogenannten BCD-Zahlen zur Verfügung. Hierbei wird jede Stelle einer Dezimalzahl getrennt behandelt, wobei jeweils 4 Bit benötigt werden:

Binär	Dezimal
0000	0
0001	1
0010	2
0011	3
0100	4

Binär	Dezimal
0101	5
0110	6
0111	7
1000	8
1001	9

Die Werte von 1010 bis 1111 (dezimal 10–15) bleiben ungenutzt. Wenn man nun eine Dezimalzahl, die größer als 9 ist, beschreiben möchte, muß man für jede Zehnerstelle vier eigene Bit vorsehen, z.B. für die Zahlen 10–15:

Binär		**Dezimal**
Zehnerstelle	Einerstelle	
0001	0000	10
0001	0001	11
0001	0010	12
0001	0011	13
0001	0100	14
0001	0101	15

Wahrscheinlich werden Sie nun nach dem Sinn der BCD-Zahlen fragen, liebe Leser, zumal z.B. die Zahl 200 nicht mehr nur 8 Bit, sondern 12 Bit für die Darstellung benötigt. Nun, es gibt einige Chips, die man mit BCD-Zahlen programmieren muß. Den C-64-Umsteigern wird vielleicht die Echtzeituhr in dem I/O-Baustein 6526 ein Begriff sein, welche die Eingaben der Stunden, Minuten und Sekunden im BCD-Format erwartet. Während der 6510 die Umschaltung in den BCD-Modus mit Hilfe des Dezimalflags durchführt, weist der 68000 eigenständige Rechenbefehle auf. Da man im BCD-Modus immer mit Überträgen rechnen muß, wird in jedem Fall das X-Flag mit einbezogen, es gibt keine übertragslosen BCD-Rechenbefehle. Es ist daher imminent wichtig, vor der ersten Rechenoperation das X-Flag zu löschen, um nicht einen zufälligen Übertrag mitzunehmen. Eine andere Besonderheit liegt im Zero-Flag: Dieses wird nämlich dann Null, wenn das Ergebnis der Rechnung nicht Null ist, bleibt aber unverändert, wenn das Ergebnis Null ist. Um das Ergebnis Null prüfen zu können, muß man deshalb vor der Rechnung das Flag auf eins setzen, wie folgendes Gegenbeispiel zeigt:

Annahme : Zero-Flag = 0 vor Beginn der Rechnung

Fall a) Ergebnis ist Null, Zero-Flag bleibt unverändert (Null)
Fall b) Ergebnis ist nicht Null, Zero-Flag wird Null

Damit wäre ein Unterschied nicht mehr auszumachen! Bei der Multiplikation zweier 16-Bit-Zahlen kann man noch zwischen vorzeichenbehafteten Zahlen (–32 768...+32 767, MULS-Befehl) und vorzeichenlosen Befehlen (0..65 535, MULU-Befehl) unterscheiden. Dies trifft auch auf die Division (DIVS, DIVU) zu. Hier tritt aber folgende Frage auf: Was passiert, wenn die Division nicht aufgeht? Das Ergebnis der Division der 32-Bit-Zahl durch die 16-Bit-Zahl wird in die unteren 16 Bit des Registers geschrieben. Unser Prozessor benutzt die freien oberen 16 Bit dazu, den Rest der Division hineinzuschreiben. Ist eine Divison also aufgegangen, erkennt man dies daran, daß die oberen 16 Bit Null sind.

Außerdem besitzt der 68000 noch einen Befehl zum Löschen aller Bits (CLR) und einen zum Negieren einer Zahl (NEG) sowie einen zur vorzeichenrichtigen Erweiterung von 8 auf 16 bzw. von 16 auf 32 Bit. Der CLR-Befehl stellt nichts anderes dar als ein MOVE-#0-Befehl, belegt aber weniger Speicher und arbeitet schneller. Bei der Negation einer Zahl wird diese von Null abgezogen. Hierbei kann wieder wahlweise mit Unterlauf durch das X-Flag gearbeitet oder im BCD-Format gerechnet werden. Der EXT-Befehl schließlich darf nur auf vorzeichenbehaftete Zahlen angewendet werden. Angenommen, Sie haben eine 8-Bit-Zahl

%10000001

und möchten diese auf 16 Bit erweitern. Es stellt sich nun die Frage, ob es sich um eine vorzeichenlose Zahl (+129) oder eine vorzeichenbehaftete Zahl (-127) handelt, bei der das siebte Bit als Vorzeichenbit genutzt wird. Der EXT-Befehl wird immer den letzteren Fall annehmen und auf

%1111111110000001

erweitern, was ebenfalls -127 entspricht. Falls Sie jedoch mit vorzeichenlosen Zahlen rechnen, hätte dieser Befehl fatale Folgen, da die Zahl 129 in die Zahl 65 409 umgewandelt worden wäre(!) Richtig würde die 129 im 16-Bit-Format als

%0000000010000001

dargestellt worden. Sie sehen, Vorsicht ist auch hier die Mutter der Porzellankiste.

Befehl: ADD ea,Dn						Funktion: Addition von ea zu einem Datenregister						
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	12	14	8
W	4	4	8	8	10	12	14	12	16	12	14	8
L	6	8	14	14	16	18	20	18	22	18	20	14
Flags	X	N	Z	V	C	Bemerkungen: Keine Berücksichtigung eines Übertrags einer vorherigen Rechenoperation						
	+	+	+	+	+							

Befehl: ADD Dn,ea						Funktion: Addition eines Datenregisters zu ea						
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*
Flags	X	N	Z	V	C	Bemerkungen: Siehe ADD ea,Dn						
	+	+	+	+	+							

Befehl: ADDA ea,An						Funktion: Addition von ea zu einem Adreßregister						
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	8	8	12	12	14	16	18	16	20	16	18	12
L	8	8	14	14	16	18	20	18	22	20	20	14
Flags	X	N	Z	V	C	Bemerkungen: Siehe ADD ea,Dn						

Bild 1.12: Die Arithmetik-Befehle ADD ea,Dn; ADD Dn,ea; ADDA ea,An

Befehl: ADDX ea,Dn	Funktion: Add. von ea und X-Flag zu Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	*	*	*	*	*	*	*	*	*	*
W	4	*	*	*	*	*	*	*	*	*	*	*
L	8	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+	+	+	+	+	Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0

Befehl: ADDX ea,-(An)	Funktion: Addition von ea und X-Flag zu -(An)

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	18	*	*	*	*	*	*	*
W	*	*	*	*	18	*	*	*	*	*	*	*
L	*	*	*	*	30	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+	+	+	+	+	Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0

Befehl: ADDI ###K,ea	Funktion: Addition einer 32-Bit-Konstanten zu ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	16	16	18	20	22	20	24	*	*	*
W	8	*	16	16	18	20	22	20	24	*	*	*
L	16	*	28	28	30	32	34	32	36	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+	+	+	+	+	Keine Berücksichtigung eines Übertrags einer vorherigen Rechenoperation

Bild 1.13: Die Arithmetik-Befehle ADDX ea,Dn; ADDX ea,-(An); ADDI ###K,ea

Befehl: ADDQ #K,ea	Funktion: Addition einer 8-Bit-Konstanten zu ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	8	12	12	14	16	18	16	20	*	*	*
L	8	8	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Keine Berücksichtigung eines Übertrags einer vorherigen Rechenoperation
	+	+	+	+	+	

Befehl: ABCD ea,Dn	Funktion: Addiere ea,x-Flag zu Dn im BCD-Format

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0
	+		+		+	

Befehl: ABCD ea,-(An)	Funktion: Addiere ea,x-Flag zu -(An) im BCD-Format

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	18	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0
	+		+		+	

Bild 1.14: Die Arithmetik-Befehle ADDQ #K,ea; ABCD ea,Dn; ABCD ea,-(An)

Befehl: SUB ea,Dn — Funktion: Subtrahiere ea von einem Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	12	14	8
W	4	4	8	8	10	12	14	12	16	12	14	8
L	6	8	14	14	16	18	20	18	22	18	20	14

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Keine Berücksichtigung eines Unterlaufs einer vorherigen Rechenoperation

Befehl: SUB Dn,ea — Funktion: Subtraktion eines Datenregisters von ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Siehe SUB ea,Dn

Befehl: SUBA ea,An — Funktion: Subtrahiere ea von einem Adreßregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	8	8	12	12	14	16	18	16	20	16	18	12
L	8	8	14	14	16	18	20	18	22	20	20	14

Flags	X	N	Z	V	C

Bemerkungen: Siehe SUB ea,Dn

Bild 1.15: Die Arithmetik-Befehle SUB ea,Dn; SUB Dn,ea; SUBA ea,An

Befehl: SUBX ea,Dn

Funktion: Subtrahiere ea und X-Flag von Datenregister

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	*	*	*	*	*	*	*	*	*	*
W	4	*	*	*	*	*	*	*	*	*	*	*
L	8	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Ergebnis = 0 : Zero-Flag bleibt
Ergebnis <>0 : Zero-Flag = 0

Befehl: SUBX ea,-(An)

Funktion: Subtrahiere ea und X-Flag von -(An)

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	18	*	*	*	*	*	*	*
W	*	*	*	*	18	*	*	*	*	*	*	*
L	*	*	*	*	30	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Ergebnis = 0 : Zero-Flag bleibt
Ergebnis <>0 : Zero-Flag = 0

Befehl: SUBI ###K,ea

Funktion: Subtraktion einer 32-Bit-Konstante von ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	16	16	18	20	22	20	24	*	*	*
W	8	*	16	16	18	20	22	20	24	*	*	*
L	16	*	28	28	30	32	34	32	36	*	*	*

Flags	X	N	Z	V	C
	+	+	+	+	+

Bemerkungen: Keine Berücksichtigung eines Unterlaufs einer vorherigen Rechenoperation

Bild 1.16: Die Arithmetik-Befehle SUBX ea,Dn; SUBX ea,-(An); SUBI ###K,ea

Befehl: SUBQ #K,ea	Funktion: Subtraktion einer 8-Bit-Konstanten von ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	8	12	12	14	16	18	16	20	*	*	*
L	8	8	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+	+	+	+	+	Keine Berücksichtigung eines Unterlaufs einer vorherigen Rechenoperation

Befehl: SBCD ea,Dn	Funktion: Subtrahiere ea,x-Flag von Dn im BCD-Format

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6	*	*	*	*	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+		+		+	Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0

Befehl: SBCD ea,-(An)	Funktion: Subtrahiere ea,x-Flag von -(An) im BCD-For.

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	18	*	*	*	*	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
	+		+		+	Ergebnis = 0 : Zero-Flag bleibt Ergebnis <>0 : Zero-Flag = 0

Bild 1.17: Die Arithmetik-Befehle SUBQ #K,ea; SBCD ea,Dn; SBCD ea,-(An)

Befehl: NEG ea	Funktion: Subtrahiere ea von Null

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: ea wird mit Ergebnis überschrieben
	+	+	+	+	+	

Befehl: NEGX ea	Funktion: Subtrahiere ea,x-Flag von Null

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: X-Flag 0/1 : Neuner-/Zehner-Komplement ea wird mit Ergebnis überschrieben
	+	+	+	+	+	

Befehl: NBCD ea	Funktion: Subtrahiere ea,x-Flag von 0 im BCD-Format

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6	*	12	12	14	16	18	16	20	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: siehe NEG ea
	+		+		+	

Bild 1.18: Die Arithmetik-Befehle NEG ea; NEGX ea; NBCD ea

Befehl: MULS ea,Dn	Funktion: Multipliziere vorzeichenbehaftet Dn mit ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	70	*	74	74	76	78	80	78	82	78	80	74
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Bei Dn nur Bits 0-15 berücksichtigt (-32768...+32767)
		+	+	0	0	

Befehl: MULU ea,Dn	Funktion: Multipliziere vorzeichenlos Dn mit ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	70	*	74	74	76	78	80	78	82	78	80	74
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Bei Dn nur Bits 0-15 berücksichtigt (0..+65535)
		+	+	0	0	

Befehl: DIVS ea,Dn	Funktion: Dividiere vorzeichenbehaftet Dn durch ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	158	*	162	162	164	166	168	166	170	166	168	162
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Ergebnis in Dn Bits 0 bis 15, eventueller Rest in Bits 16 bis 31
		+	+	+	0	

Bild 1.19: Die Arithmetik-Befehle MULS ea,DN; MULU es,Dn; DIVS ea,Dn

Befehl: DIVU ea,Dn	Funktion: Dividiere vorzeichenlos Dn durch ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	140	*	144	144	146	148	150	148	152	148	150	144
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Ergebnis in Dn Bits 0 bis 15, eventueller Rest in Bits 16 bis 31
		+	+	+	0	

Befehl: CLR ea	Funktion: Alle Bits von ea werden auf Null gesetzt.

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Keine
		0	1	0	0	

Befehl: EXT ea	Funktion: Erweiterung von 8/16 auf 16/32 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	4	*	*	*	*	*	*	*	*	*	*	*
L	4	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Zahl wird als vorzeichenbehaftet angenommen, daher fehlerhaft bei vorzeichenlosen Zahlen
		+	+	0	0	

Bild 1.20: Die Arithmetik-Befehle DIVU ea, Dn; CLR ea; EXT ea

1.3.2.3 Die logischen Befehle

Durch die logischen Befehle wird eine bitweise Verknüpfung zwischen Quell- und Zieloperand durchgeführt. Während die Befehle AND, OR und EOR eine Verknüpfung zwischen einem Quell- und einem Zieloperanden durchführen und sich darin unterscheiden, welches Ergebnis sie den möglichen vier Verknüpfungen zuweisen, wird durch den NOT-Befehl das jeweilige Bit einfach invertiert z.B.

```
NOT %10010110 = %01101001
```

Hier nun eine Übersicht über die Wirkung der übrigen drei Verknüpfungen:

Verknüpfung	AND	OR	EOR
0 verknüpft mit 0	0	0	0
0 verknüpft mit 1	0	1	1
1 verknüpft mit 0	0	1	1
1 verknüpft mit 1	1	1	0

Während die AND-Verknüpfung (»Logisch-Und«) nur dann ein »1«-Ergebnis liefert, wenn beide Bits eins sind, geschieht dies bei der OR-Verknüpfung (»Logisch-Oder«), wenn mindestens ein Bit eins ist und bei der EOR-Verknüpfung (»Exklusiv-Oder«), wenn genau ein Bit eins ist. Ein Beispiel für das unterschiedliche Ergebnis einer 16-Bit-Verknüpfung:

```
    %1100111010010110        %1100111010010110        %1100111010010110
AND %1011001101111011     OR %1011001101111011    EOR %1011001101111011
---------------------------------------------------------------------------
    %1000001000010010        %1111111111111111        %0111110111101101
```

1.3.2.4 Die Verschiebebefehle

Neben den arithmetischen und logischen Befehlen kennt unsere CPU noch eine weitere Gruppe von Befehlen, mit denen Daten verarbeitet werden können, die sogenannten Schiebe- und Rotationsbefehle. Bei beiden Befehlsgruppen wird ein Byte, Wort oder Langwort bitweise nach links bzw. rechts verschoben. Der Unterschied zum 6510 liegt u.a. darin, daß mit einem Befehl mehrmals verschoben/rotiert werden kann. Die prinzipielle Differenz zwischen den Rotations- und Verschiebebefehlen liegt darin, daß bei den Rotationsbefehlen - wie der Name es schon ausdrückt - das hinausgeschobene Bit auf der anderen Seite wieder eingespeist wird, während bei den Verschiebebefehlen entweder eine Null eingesetzt wird oder das vor der Verschiebung äußerste Bit wieder an diesen Platz kopiert wird.

Befehl: AND ea,Dn													Funktion: Logisch-Und zwischen ea und Datenregister
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#	
B	4	*	8	8	10	12	14	12	16	12	14	8	
W	4	*	8	8	10	12	14	12	16	12	14	8	
L	6	*	14	14	16	18	20	18	22	18	20	14	

Flags	X	N	Z	V	C	Bemerkungen: Keine
		+	+	0	0	

Befehl: AND Dn,ea													Funktion: Logisch-Und zwischen Datenregister und ea
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#	
B	4	*	12	12	14	16	18	16	20	*	*	*	
W	4	*	12	12	14	16	18	16	20	*	*	*	
L	6	*	20	20	22	24	26	24	28	*	*	*	

Flags	X	N	Z	V	C	Bemerkungen: Keine
		+	+	0	0	

Befehl: ANDI ###K,ea												Funktion: Logisch-Und zwischen 32-Bit-Konstante u.ea
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	CCR	SR	
B	8	*	16	16	18	20	22	20	24	20	*	
W	8	*	16	16	18	20	22	20	24	*	20 (P)	
L	16	*	28	28	30	32	34	32	36	*	*	

Flags	X	N	Z	V	C	Bemerkungen: Bei Zugriff auf das Statusregister Privilegierter Befehl (P).
		+	+	0	0	

Bild 1.21: Die logischen Befehle AND ea,Dn; AND Dn,ea; ANDI ###K,ea

Befehl: OR ea,Dn	Funktion: Logisch-Oder zwischen ea und Datenregister											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	12	14	8
W	4	*	8	8	10	12	14	12	16	12	14	8
L	6	*	14	14	16	18	20	18	22	18	20	14
Flags X N Z V C + + 0 0	Bemerkungen: Keine											

Befehl: OR Dn,ea	Funktion: Logisch-Oder zwischen Datenregister und ea											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*
Flags X N Z V C + + 0 0	Bemerkungen: Keine											

Befehl: ORI ###K,ea	Funktion:Logisch-Oder zwischen 32-Bit-Konstante u.ea										
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	CCR	SR
B	8	*	16	16	18	20	22	20	24	20	*
W	8	*	16	16	18	20	22	20	24	*	20 (P)
L	16	*	28	28	30	32	34	32	36	*	*
Flags X N Z V C + + 0 0	Bemerkungen: Bei Zugriff auf das Statusregister Privilegierter Befehl (P).										

Bild 1.22: Die logischen Befehle OR ea,Dn; OR Dn,ea; ORI ###K,ea

Befehl: EOR Dn,ea	Funktion:Exklusiv-Oder zwischen Datenregister und ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Keine
		+	+	0	0	

Befehl: EORI ###K,ea	Funktion:Exkl.-Oder zwischen 32-Bit-Konstante u. ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	CCR	SR
B	8	*	16	16	18	20	22	20	24	20	*
W	8	*	16	16	18	20	22	20	24	*	20 (P)
L	16	*	28	28	30	32	34	32	36	*	*

Flags	X	N	Z	V	C	Bemerkungen: Bei Zugriff auf das Statusregister Privilegierter Befehl (P).
		+	+	0	0	

Befehl: NOT ea	Funktion: Logisch-Nicht von ea

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	12	12	14	16	18	16	20	*	*	*
W	4	*	12	12	14	16	18	16	20	*	*	*
L	6	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: ea wird mit dem Ergebnis überschrieben
		+	+	0	0	

Bild 1.23: Die logischen Befehle EOR Dn,ea; EORI ###K,ea; NOT ea

Zum besseren Verständnis zeigt nachfolgende Übersicht, was bei den einzelnen Verschiebebefehlen mit den jeweiligen Bits passiert, hier als Beispiel für die Wort-Adressierung:

Befehl	Schieberichtung	Bit 15 nach	Bit 0 nach	nach Bit 15	nach Bit 0
LSL	links	C/X-Flag	Bitl	Bit 14	0
LSR	rechts	Bit 14	C/X-Flag	0	Bit 1ftl
ASL	links	C/X-Flag	Bit 1	Bit 14	0ftl
ASR	rechts	Bit 14	C/X-Flag	Bit 15	Bit 1

Bei den Rotationsbefehlen muß man zwischen den »echten« Rotierern, die das herausgeschobene Bit auf der anderen Seite wieder einblenden, und den Übertrags-Rotierern unterscheiden. Jene Befehle blenden nämlich einerseits das X-Flag in die freigewordene Bitposition ein, schieben auf der anderen Seite aber auch das hinausgeworfene Bit in dieses Flag, so daß ein mehrmaliger Aufruf dieser Befehle wieder zu einer echten Rotation führen würde. Die Bedeutung dieser Version liegt darin, daß man so auch mehr als 32 Bit in 32-Bit-Gruppen verschieben kann. Auch hier soll eine Übersicht die Funktion der Befehle verdeutlichen, diesmal bei einer Langwortadressierung :

Befehl	Rotationsrichtung	Bit 31 nach	Bit 0 nach	nach Bit 31	nach Bit 0
ROL	links	C-Flag, Bit 0	Bit 1	Bit 30	Bit 31
ROR	rechts	Bit 30	C-Flag, Bit 31	Bit 0	Bit 1
ROXL	links	C/X-Flag	Bit 1	Bit 30	X-Flag
ROXR	rechts	Bit 30	C/X-Flag	X-Flag	Bit 1

Anfangs scheint man etwas ratlos zu sein, was man mit diesen Befehlen überhaupt anfangen kann. Ein Beispiel soll daher dieses Rätsel lösen. An dem Wert $46 wollen wir zunächst den LSL- und dann den LSR-Befehl ausprobieren:

```
$46 = %010000110 - LSL - %100001100 = $8C, Carry = 0
$46 = %010000110 - LSR - %001000011 = $23, Carry = 0
```

Man erkennt, daß der Wert durch den Einsatz des LSL-Befehls verdoppelt wurde, während er durch den LSR-Befehl halbiert wurde. Dies ist ja ganz logisch, da ja im Binärsystem die folgende Stelle immer den doppelten Wert innehat.

Man hat eine einfache Möglichkeit gefunden, Multiplikationen und Divisionen mit Faktoren durchzuführen, die durch zwei teilbar sind. Wenn man z.B. um 3 Bit nach links verschiebt, wäre das Ergebnis achtmal so groß (8 = 2*2*2).

Wozu, werden Sie nun fragen, haben wir überhaupt den MULU- bzw. MULS-Befehl, wenn wir sie scheinbar doch nicht benutzen? Man muß hier eindeutig differenzieren: Fest steht, daß die Verschiebebefehle wesentlich schneller arbeiten als die Multiplikationsbefehle, so daß man sie immer dann auch auf 32-Bit-Operanden ansetzen sollte, wenn der

Multiplikationsfaktor eine durch zwei teilbare Zahl ist. Auf der anderen Seite lohnt es sich nicht, Multiplikationen mit Faktoren wie z.B.

14 = 2*2*2 + 2*2 + 2

durch dreimaligen Aufruf eines Verschiebebefehls durchzuführen. Hier muß man immer abwägen, was denn im speziellen Fall günstiger ist. Alles eben Gesagte gilt natürlich auch für die Divisionen. Hier darf man allerdings nur dann die Verschiebebefehle benutzen, wenn man an einem etwaigen Rest der Division nicht interessiert ist.

Richtig aufregend wird die Benutzung dieser Befehle aber erst, wenn die Operanden für die Multiplikations- und Divisionsbefehle zu groß sind und diese damit versagen. Nehmen wir einmal an, wir möchten die 64-Bit-Zahl

%00000011000110111011010000110111 10110011010011010010110101010101

mit dem Faktor 2 multiplizieren. Hier kann der MULU-Befehl nicht angewendet werden, dafür aber diese:

```
move.l %10110011010011010010110101010101,d0 ;Niederwertige 32 Bit laden
move.l %00000011000110111011010000110111,d1 ;Höherwertige 32 Bit laden
LSL d0                                      ;Niederwertige 32 Bit verschieben
ROXL d1                                     ;Höherwertige 32 Bit verschieben
```

Nach dem LSL-Befehl steht das herausgeschobene 1-Bit im X-Flag zur Verfügung, von wo es durch den ROXL-Befehl an die unterste Position der höherwertigen 32 Bit eingeblendet wird. Analog lassen sich auch 64-Bit-Divisionen mit den Befehlen LSR und ROXR durchführen.

Befehl: ASL ea,Dn						Funktion: Linksverschiebung um (Inhalt von ea) Bits						
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x
Flags	X	N	Z	V	C	Bemerkungen: x: Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, MSB nach X/C-Flag, 0 nach LSB						
	+	+	+	+	+							

Bild 1.24: Der Verschiebebefehl ASL ea,Dn

Befehl: ASL ea	Funktion: Linksverschiebung um 1 Bit											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*
Flags X N Z V C + + + + +	Bemerkungen: MSB nach C/X-Flag, 0 nach LSB											

Befehl: ASR ea,Dn	Funktion: Rechtsverschiebung um (Inhalt von ea) Bits											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6*12	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x
Flags X N Z V C + + + + +	Bemerkungen: x : Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, MSB nach MSB, LSB nach C/X-Flag											

Befehl: ASR ea	Funktion: Rechtsverschiebung um 1 Bit											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L			
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*
Flags X N Z V C + + + + +	Bemerkungen: MSB nach MSB, LSB nach C/X-Flag											

Bild 1.25: Die Verschiebebefehle ASL ea; ASR ea,Dn; ASR ea

Befehl: LSL ea,Dn	Funktion: Linksverschiebung um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen: siehe ASL ea,Dn. Beide Befehle arbeiten identisch (außer V-Flag)
	+	+	+	0	+	

Befehl: LSL ea	Funktion: Linksverschiebung um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: siehe ASL ea. Beide Befehle arbeiten identisch (außer V-Flag)
	+	+	+	0	+	

Befehl: LSR ea,Dn	Funktion: Rechtsverschiebung um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen: x: Anzahl (1-8) der zu verschiebenden Bits= Inhalt von ea, LSB nach C/X-Flag, 0 nach MSB
	+	+	+	0	+	

Bild 1.26: Die Verschiebebefehle LSL ea,Dn; LSL ea; LSR ea,Dn

Befehl: LSR ea	Funktion: Rechtsverschiebung um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: 0 nach MSB, LSB nach C/X-Flag
	+	+	+	0	+	

Befehl: ROL ea,Dn	Funktion: Linksrotieren um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen: x: Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, MSB nach LSB und C-Flag
		+	+	0	+	

Befehl: ROL ea	Funktion: Linksrotieren um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: MSB nach LSB und C-Flag
		+	+	0	+	

Bild 1.27: Die Verschiebe- und Rotationsbefehle LSR ea; ROL ea,Dn; ROL ea

Befehl: ROR ea,Dn	Funktion: Rechtsrotieren um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen:
		+	+	0	+	x: Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, LSB nach MSB und C-Flag

Befehl: ROR ea	Funktion: Rechtsrotieren um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	6+2x
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen:
		+	+	0	+	LSB nach MSB und C-Flag

Befehl: ROXL ea,Dn	Funktion: Linksrotieren um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen:
		+	+	0	+	x: Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, MSB nach C/X-, X-Flag nach LSB

Bild 1.28: Die Rotationsbefehle ROR ea, DN; ROR ea,Dn; ROXL ea, Dn

Befehl: ROXL ea	Funktion: Linksrotieren um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: MSB nach C/X-Flag, X-Flag nach LSB
	+	+	+	0	+	

Befehl: ROXR ea,Dn	Funktion: Rechtsrotieren um (Inhalt von ea) Bits

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
W	6+2x	*	*	*	*	*	*	*	*	*	*	6+2x
L	8+2x	*	*	*	*	*	*	*	*	*	*	8+2x

Flags	X	N	Z	V	C	Bemerkungen: x: Anzahl (1-8) der zu verschiebenden Bits = Inhalt von ea, LSB nach C/X-, X-Flag nach MSB
	+	+	+	0	+	

Befehl: ROXR ea	Funktion: Rechtsrotieren um 1 Bit

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	*	*	12	12	14	16	18	16	20	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: LSB nach C/X-Flags, X-Flag nach MSB
	+	+	+	0	+	

Bild 1.29: Die Rotationsbefehle ROXL ea; ROXR ea,Dn; ROXR ea

1.3.2.5 Die Bit-Befehle

Die Umsteiger vom C64 unter Ihnen, liebe Leser, werden wie ich mit Grauen daran zurückdenken, wie umständlich es war, auf ein bestimmtes Bit Einfluß zu nehmen. Da der 6510 keine Befehle hierfür bereitstellt, mußte man sich immer wieder mit einem Trick, dem sogenannten »Ausmaskieren« mit Hilfe logischer Befehle helfen. Zum Glück sieht die Sache beim 68000 doch sehr viel freundlicher aus: er besitzt drei Befehle mit denen man ein einziges Bit beeinflussen kann:

BCHG : Zustand eines Bits ändern (aus Eins wird Null und umgedreht)
BCLR : Bit löschen (auf Null setzen)
BSET : Bit setzen (auf Eins setzen)

Dabei wird der vorhergehende Zustand des Bits in das Zero-Flag übertragen (Zero-Flag ist also gesetzt falls dieses Bit 0 war), so daß man ihn bequem abfragen kann (siehe übernächster Abschnitt). Für den Fall, daß man den Zustand eines Bits nur abfragen, aber nicht ändern möchte, existiert ein weiterer Befehl:

BTST : Zustand eines Bits testen

Falls die Bit-Befehle auf ein Datenregister angewendet werden, können die Bits 0 bis 31 bearbeitet werden, bei einer anderen effektiven Adresse nur die Bits 0 bis 7. Daraus folgt, daß man z.B. für den Fall, das zwölfte Bit eines Wortes im Speicher testen zu wollen, die Adresse modifizieren muß. Statt

```
BTST #12,(A0)     ;illegaler Befehl, existiert nicht!
```

könnte man z.B. schreiben

```
BTST #4,(-A0)
```

Hier würde zunächst der Adreßpointer um 1 Byte (8 Bit) erniedrigt, so daß das höherwertige Byte des Wortes verarbeitet werden kann. Neben diesen »lupenreinen« Bit-Befehlen weist der 68000 noch einen Sonderbefehl auf, den man mit dem BIT-Befehl des 6510 vergleichen kann, den TAS-Befehl. Er testet die gesamte effektive Adresse und setzt abhängig davon das Negativ- und Zero-Flag. Anschließend wird das siebte Bit auf eins gesetzt. Diesen Befehl kann man ausschließlich in der Byte-Adressierung verwenden, selbst wenn auf ein Datenregister zugegriffen wird.

Befehl: BCHG Dn,ea — Funktion: Bit (Inhalt von Dn) von ea ändern

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	12	12	14	16	18	16	20	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	8	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
			+		

Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea<>Dn werden die Bits 8-31 von Dn ignoriert.

Befehl: BCHG ###K,ea — Funktion: Bit-Nr. ###K von ea ändern

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	16	16	18	20	22	20	24	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	12	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
			+		

Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea=Dn 0<=K<=31, sonst 0<=K<=7

Befehl: BCLR Dn,ea — Funktion: Bit (Inhalt von Dn) von ea löschen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	12	12	14	16	18	16	20	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	8	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
			+		

Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea<>Dn werden die Bits 8-31 von Dn ignoriert.

Bild 1.30: Die Bit-Befehle BCHG Dn,ea; BCHG ###K,ea; BCLR Dn,ea

Befehl: BCLR ###K,ea	Funktion: Bit-Nr. ###K von ea löschen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	16	16	18	20	22	20	24	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	12	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea=Dn 0<=K<=31, sonst 0<=K<=7
			+			

Befehl: BSET Dn,ea	Funktion: Bit (Inhalt von Dn) von ea setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	12	12	14	16	18	16	20	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	8	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea<>Dn werden die Bits 8-31 von Dn ignoriert.
			+			

Befehl: BSET ###K,ea	Funktion: Bit-Nr. ###K von ea setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	16	16	18	20	22	20	24	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	12	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Vorheriger Zustand ins Z-Flag. Bei ea=Dn 0<=K<=31, sonst 0<=K<=7
			+			

Bild 1.31: Die Bit-Befehle BCLR #KKK,ea; BSET Dn,ea; BSET ###K,ea

Befehl: BTST Dn,ea — Funktion: Bit (Inhalt von Dn) von ea testen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	8	8	10	12	14	12	16	12	14	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	6	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
			+		

Bemerkungen: Zustand kommt in das Z-Flag. Bei ea<>Dn werden die Bits 8-31 von Dn ignoriert.

Befehl: BTST ###K,ea — Funktion: Bit-Nr. ###K von ea testen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	12	12	14	16	18	16	20	16	18	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	10	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
			+		

Bemerkungen: Zustand kommt in das Z-Flag. Bei ea=Dn 0<=K<=31, sonst 0<=K<=7

Befehl: TAS ea — Funktion: ea testen und Bit 7 setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	14	14	16	18	20	18	22	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C
		+	+	0	0

Bemerkungen: Vorzeichentest (N-Flag) und auf Null (Z-Flag)

Bild 1.32: Die Bit-Befehle BTST Dn,ea; BTST ###K,ea; TAS ea

1.3.2.6 Die Vergleichsbefehle

Unverzichtbar für jedes Programm sind die Vergleichsbefehle, da man nur durch ihre Verwendung Entscheidungen treffen kann. Die CMP-Befehle des 68000 wirken intern so, daß der Quelloperand vom Zieloperand subtrahiert wird und die Flags wie beim SUB-Befehl gesetzt werden. Der einzige Unterschied liegt darin, daß der Zieloperand nicht mit dem Ergebnis beschrieben wird.

Ein spezieller Vergleichsbefehl - TST - vergleicht einen Operanden mit der Zahl Null, indem er von dieser abgezogen wird. Wie man die Ergebnisse der Vergleichsbefehle, die Flagzustände, auswerten kann, werden wir uns im nächsten Abschnitt ansehen.

Befehl: CMP ea,Dn	Funktion: ea von Dn abziehen ohne Ergebnis zu setzen											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	12	14	8
W	4	4	8	8	10	12	14	12	16	12	14	8
L	6	6	14	14	16	18	20	18	22	18	20	14
Flags	X	N	Z	V	C	Bemerkungen: Wirkung auf Flags wie der SUB-Befehl						
		+	+	+	+							

Befehl: CMPA ea,An	Funktion: ea von An abziehen ohne Ergebnis zu setzen											
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	*	*	*	*	*	*	*	*	*
W	6	6	10	10	12	14	16	14	18	14	16	8
L	6	6	14	14	16	18	20	18	22	18	20	18
Flags	X	N	Z	V	C	Bemerkungen: Wirkung auf Flags wie der SUBA-Befehl						
		+	+	+	+							

Bild 1.33: Die Vergleichsbefehle CMP ea,Dn und CMPA ea,An

Befehl: CMPI ###K,ea	Funktion: K von ea abziehen ohne Ergebnis zu setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	8	*	12	12	14	16	18	16	20	*	*	*
W	8	*	12	12	14	16	18	16	20	*	*	*
L	14	*	20	20	22	24	26	24	28	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Wirkung auf Flags wie der SUBI-Befehl
		+	+	+	+	

Befehl: CMPM (An)+,ea	Funktion:(An) von ea abziehen ohne Ergebnis zu setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	*	*	*	12	*	*	*	*	*	*	*	*
W	*	*	*	12	*	*	*	*	*	*	*	*
L	*	*	*	20	*	*	*	*	*	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Wirkung auf Flags wie der SUB-Befehl
		+	+	+	+	

Befehl: TST ea	Funktion: ea von 0 abziehen ohne Ergebnis zu setzen

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	4	*	8	8	10	12	14	12	16	*	*	*
W	4	*	8	8	10	12	14	12	16	*	*	*
L	4	*	12	12	14	16	18	16	20	*	*	*

Flags	X	N	Z	V	C	Bemerkungen: Vorzeichentest (N-Flag) und auf Null (Z-Flag)
		+	+	0	0	

Bild 1.34: Die Vergleichsbefehle CMPI ###K,ea; CMPM (An)+,ea; TST ea

1.3.2.7 Die Befehle zur bedingten Verzweigung

Um die Zustände der Flags im Programm ausnutzen zu können, gibt es Befehle, mit deren Hilfe man Entscheidungen treffen kann. Damit kann man z.B. auf die Vergleichsbefehle reagieren. Je nach Zustand einzelner Flags oder einer Kombination von mehreren Flags kann man auf das Verhältnis der verglichenen Operanden schließen. In Bild 1.36 sehen Sie die Bedingungscodes der drei zur Verfügung stehenden Befehle Bcc, DBcc und Scc aufgelistet. »cc« steht für Condition Code und muß im Programm durch den Code für die jeweilige Bedingung ersetzt werden z.B.

```
BCS Label   ;verzweigt bei gesetztem Carry-Flag
```

Die in Bild 1.35 mit einem Stern gekennzeichneten Bedingungscodes sind für die Anwendung mit vorzeichenbehafteten Zahlen reserviert, die übrigen für vorzeichenlose. Damit der Prozessor weiß, wohin er springen muß, wäre es normalerweise erforderlich, hinter dem Befehlscode eine 32-Bit-Adresse anzugeben. Der Nachteil dieser Adressierung wäre aber darin zu sehen, daß die Sprungadresse bei jeder Speicherverschiebung des Programms mitgeändert werden müßte. Da man in der Praxis sehr viele Sprünge dieser Art benötigt, hat man sich die relative Adressierung einfallen lassen. Hierbei folgt hinter dem Befehlscode ein Offset, der den Abstand der Zieladresse vom augenblicklichen Stand des Programmzählers angibt. Diese Methode erlaubt ein beliebiges Verschieben des Programms im Speicher, da sich der Offset ja nicht ändert. Für ihn wurden 16 Bit vorgesehen, so daß man nur im Rahmen von 65 526 Byte springen kann. Um auch Rückwärtssprünge zu ermöglichen, werden die Offsets, die größer als 32 767 sind, als Zweierkomplement von negativen Zahlen interpretiert. Somit kann man um 32 768 Byte zurück- und um 32 767 Byte vorwärtsspringen.

Der Bcc-Befehl dient wie beim 6510 ausschließlich dazu, das Ziel anzuspringen. Sehr viel komplizierter arbeitet der Befehl

```
DBcc Dn,Label
```

Falls cc erfüllt ist, wird einfach mit dem nächsten Befehl fortgefahren. Sonst wird das Register Dn um eins vermindert und geprüft, ob es auf −1 heruntergezählt wurde. Falls dies eingetroffen ist, wird der Befehl ebenfalls übersprungen, sonst wird zum angegebenen Label gesprungen. Was bringt uns dieser Befehl in der Praxis? Nun, er ist als Zähler wie geschaffen. Angenommen, Sie möchten zehn Langwörter von einer Adresse (in A0) zu einer anderen (in A1) kopieren. In diesem Fall müssen Sie zehnmal den Befehl

```
move.l (a0)+,(a1)+
```

aufrufen. Sie lassen nun einfach ein Datenregister von 10 auf 0 herunterzählen. Solange der Wert Null nicht erreicht wird, springen Sie einfach wieder die Move-Zeile an:

```
      move.l #10,d0
loop  move.l (a0)+,(a1)+
      dbne d0,loop
```

Der Scc-Befehl schließlich setzt alle Bits der effektiven Adresse, wenn cc erfüllt ist, sonst werden die Bits gelöscht.

cc (condition Code)	Bedeutung	Test auf	Bcc	Dbcc	Scc
HI	high	>			
LS	low or same	<=			
CC	carry clear	C-Flag=0			
CS	carry set	C-Flag=1			
NE	not equal	<>			
EQ	equal	=			
VC	overflow clear	V-Flag=0	*	*	*
VS	overflow set	V-Flag=1	*	*	*
PL	plus	> 0			
MI	minus	< 0			
GE	greater or equal	>=		*	*
LT	less than	<	*	*	*
GT	greater than	>	*	*	*
LE	less or equal	<=	*	*	*
F	false	immer false		*	*
T	true	immer true		*	*

Bild-1.35: Die Bedingungscodes

```
Befehl: Bcc label          Funktion: Falls cc erfüllt : PC = PC + D
Zyklen : Bei D= 8/16 Bit : cc erfüllt, 10/10, sonst 8/12
```

```
Befehl: DBcc Dn,Label
Funktion : cc erfüllt : PC = PC + 2
                sonst : Falls Dn<>-1: PC = PC + D
                              sonst: PC = PC + 2
```

Befehl: Scc ea												
Funktion: Falls cc erfüllt: ea=$FF, sonst ea=$00												
ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
B	6/4[1]	*	12	12	14	16	18	16	20	*	*	*
W	*	*	*	*	*	*	*	*	*	*	*	*
L	*	*	*	*	*	*	*	*	*	*	*	*
Flags X N Z V C												
Bemerkungen: [1]: Falls cc erfüllt, 4 Zyklen												

Bild 1.36: Die Befehle zur bedingten Verzweigung Bcc, DBcc und Scc

1.3.2.8 Die Unterprogramm- und Sprungbefehle

Für den Ansprung einer Adresse und eines Unterprogramms hält unser Prozessor jeweils zwei Befehlsversionen bereit: Die eine arbeitet wie die Befehle zur bedingten Verzweigung, indem die Zieladresse durch einen relativen 16-Bit-Offset zum jeweiligen Programmzählerstand bestimmt wird, die andere durch die absolute Angabe der Adresse:

Sprung in	Unterprogramm	Hauptprogramm
relativ	BSR	BRA
absolut	JSR	JMP

Der Vorteil der letzten Methode besteht darin, daß der Abstand des Sprungziels von dem Sprungaufruf beliebig groß sein darf und daß mehrere Adressierungsarten zur Verfügung stehen. Die Befehle benötigen aber nicht nur mehr Platz, weil die Adreßangabe 1 Langwort benötigt, sondern arbeiten auch langsamer als die Kollegen mit der relativen Adressierung.

Die Befehle zum Ansprung eines Unterprogramms schieben vor dem Sprung noch schnell die Rücksprungadresse auf den Stapel, damit nach der Beendigung des Unterprogramms an der ursprünglichen Stelle fortgefahren werden kann. Zur Beendigung eines Unterprogramms existiert daher ein weiterer Befehl (RTS), der die Rücksprungadresse vom Stapel holt und in den Programmzähler lädt.

Schon so mancher Maschinenprogrammierer ist daran zu Grunde gegangen, daß er den Stack innerhalb eines Unterprogramms geändert und damit die korrekte Rücksprungadresse eliminiert hat. Die Folge ist meistens ein Adreß-Error und damit eine der beliebten....na, Sie wissen schon!

Befehl: BRA label	Funktion: 8/16-Bit-Offset zu Programmzähler addieren
Zyklen : Bei D= 8/16 Bit : cc erfüllt, 10/10, sonst 8/12	

Befehl: JMP ea	Funktion: Programmzähler mit Zieladresse laden

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
	*	*	8	*	*	10	14	10	12	10	14	*

Befehl: BSR label	Funktion: PC nach -(SP), 16-Bit-Offset zu PC addier.

Befehl: JSR ea	Funktion: PC nach -(SP), PC mit Zieladresse laden

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
	*	*	16	*	*	18	22	18	20	18	22	*

Befehl: RTS	Funktion: Programmzähler mit (SP)+ laden

Bild 1.37: Die Sprung- und Unterprogrammbefehle BRA label, JMP ea, BSR label, JSR ea und RTS

1.3.2.9 Die übrigen 68000-Befehle

Als letzte Gruppe haben wir alle die Befehle, die sich nicht in einer der vorhergehenden einordnen ließen. Die meisten von ihnen sind sowieso für den Normalverbraucher nicht so interessant, da sie eine Exception und bei unveränderten Vektoren eine Guru-Meditation auslösen, was ja nicht das Ziel eines Programms sein kann, oder?

Beginnen wir mit dem harmlosesten, dem NOP-Befehl. Seine Wirkung besteht wie beim 6510 im Nichtstun. Dafür benötigt er zwei Byte an Speicher und vier Taktzyklen an Rechenzeit. Durch die CHK-Anweisung wird der Inhalt einer Adresse darauf geprüft, ob er im Bereich von 0 bis zum Inhalt eines Datenregisters liegt. Wenn dies der Fall ist, passiert gar nichts, sonst wird ein TRAP #6 mit einer Exception als Folge ausgelöst.

Die TRAP-Anweisung löst eine Exception ohne Vorbedingung aus, während der TRAPV-Befehl dies nur bei gesetztem Overflow-Flag tut. Schließlich haben wir noch zwei Befehle, um eine Exception zu beenden. Sie heißen RTE und RTR, wobei der Unterschied nur darin liegt, daß RTE privilegiert ist, während RTR auch im User-Modus ausgeführt werden darf. Wie beim RTS-Befehl wird die neue Adresse des Programmzählers vom Stapel geholt, zusätzlich aber noch das bei der Auslösung einer Exception gerettete Statusregister (RTE) bzw. Condition-Code-Register (RTR).

Zum Schluß stellt sich noch die Frage, was passiert, wenn der Prozessor auf einen Befehlscode trifft, den er nicht kennt? Die Umsteiger vom C64 haben sicherlich schon von den sogenannten »illegalen Opcodes« des 6510 gehört, die durchaus sinnvolle Operationen durchführen und von der Software-Branche dazu genutzt wurden, Programme unlesbar zu machen. Nun, beim 68000 gibt es so etwas nicht. Tritt ein illegaler Befehl auf, werden kurz und schmerzlos der Programmzähler und das Statusregister gerettet, bevor der TRAP #4-Befehl ausgeführt wird. Sie sehen, auch hier steht normalerweise ein Guru am Ende, so daß es sich wirklich nicht lohnt, mit illegalen Codes herumzuexperimentieren.

Befehl: CHK ea,Dn	Funktion: Auf 0<ea<Dn prüfen, wenn nein, TRAP #6

ea	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	$.W	$.L	d(PC)	d(PC,Rn)	#
	10	*	14	14	16	18	20	18	22	18	20	14
Trap	44	*	48	48	50	52	54	52	56	52	54	48

Flags	X	N	Z	V	C	Bemerkungen:
		+				Falls TRAP #6-Vektor unverändert, folgt Guru-Meditation

Befehl: NOP	Funktion: Programmzähler um zwei erhöhen
Zyklen : 4	

Befehl: TRAP #K	Funktion: Trap-Vektoren #K (0-15) ausführen
Zyklen : 34	Bemerkungen: 1.) PC nach -(SSP) 2.) SR nach -(SSP) 3.) Trap-Vektor nach PC

Befehl: TRAPV	Funktion: Falls V-Flag=1, Trapv-Vektor ausführen
Zyklen : 4, bei Trap-ausführung 34	Bemerkungen: 1.) PC nach -(SSP) 2.) SR nach -(SSP) 3.) Exception-Vektor 7 nach PC

Befehl: RTE	Funktion: Rückkehr von Exception
Flags X N Z V C + + + + +	Bemerkungen: 1.) (SP)+ nach SR 2.) (SP)+ nach PC Privilegierter Befehl

Befehl: RTR	Funktion: Rückkehr von Exception
Flags X N Z V C + + + + +	Bemerkungen: 1.) (SP)+ nach CCR 2.) (SP)+ nach PC

Befehl: ILLEGAL	Funktion: Exception durch illegalen Befehlscode
Zyklen : 34	Bemerkungen: 1.) PC nach -(SSP) 2.) SR nach -(SSP) 3.) TRAP #4-Vektor nach PC

Bild 1.38: Die Befehle CHK, NOP, TRAP #K, TRAPV, RTE, RTR und ILLEGAL

1.4 Die Libraries

Wenn Sie von einem Rechner wie dem C64 auf den Amiga umgestiegen sind, liebe Leser, kennen Sie sicherlich das Phänomen, daß viele Programme nicht mehr liefen, wenn man ein anderes Betriebssystem verwendete, um z.B. in den Genuß eines Floppyspeeders zu kommen. Der Grund hierfür ist in den geänderten Adressen der Betriebssystem-Einsprünge zu sehen. Die Unterprogramme zur Ausgabe von Zeichen etc. wurden in der Form »JSR XXXX« aufgerufen, wobei XXXX die absolute Adresse der Routine ist. Leider kann dieses Verfahren aber nur dann funktionieren, wenn XXXX über das gesamte Computerleben konstant bleibt, was den Schluß nahelegt, daß das Verfahren mit den Absolut-Sprüngen nicht gerade vorbildlich ist.

Beim Amiga ging man hingegen einen zukunftssicheren Weg. Alle Unterprogramme, die von dem Programmierer benutzt werden dürfen, wurden in sogenannten Libraries abgelegt, einer Sammlung von Unterprogrammen mit einer zugehörigen Adressentabelle. Für jeden

Zweck (Grafik, Fließkommarechnung, DOS, Intuition, etc.) existiert eine eigene Library. Der entscheidende Vorteil in diesem Verfahren ist darin zu sehen, daß es bis auf eine Ausnahme (EXEC-Library) keine festen Adressen gibt. Wenn man eine Funktion aus einer bestimmten Library benutzen möchte, muß man sie öffnen lassen. Dies geschieht durch ein Unterprogramm der EXEC-Library, die selbst nicht geöffnet werden muß. Diese Funktion gibt einen Zeiger auf die Startadresse der jeweiligen Tabelle zurück, in der die Adressen der einzelnen Unterprogramme gespeichert sind. Falls sich die Lage eines Unterprogramms ändern sollte, braucht man nur den Eintrag in der Tabelle zu ändern, ohne daß der Zeiger auf die entsprechende Tabellenposition geändert werden müßte, alle Programme laufen also wie gehabt. Um ein Unterprogramm benutzen zu können, muß man daher nur noch die Differenz der Startadresse der Tabelle und dem zugehörigen Tabellenplatz kennen (Offset), die beim Amiga ausnahmslos negativ sind, d.h., die Unterprogrammadressen werden vom Tabellenstart im Adreßraum nach unten abgelegt. Natürlich kann man auch einfach neue Funktionen hinzufügen, indem man die Tabelle um einen oder mehrere Einträge erweitert. Sie sehen ein, liebe Leser, daß dieses Konzept absolut zukunftssicher ist, und keine Kompatibilitätsprobleme auftreten werden, wie es der Chefredakteur einer PC-Zeitschrift seinen Lesern Monat für Monat einzureden versucht, offenbar um ein Umsteigen von dem veralteten PC auf den modernen Amiga zu verhindern.

Das einzige Problem bei der Verwendung der Libraries besteht darin, daß man sich die Offsets von mehreren Unterprogrammen merken müßte. Hier hilft uns aber unser DEVPAC-Assembler aus der Patsche: Auf der Diskette ist nämlich für jede Library ein sogenanntes Include-File mit allen Offsets vorhanden. Ein Include-File ist nichts anderes als ein Stück Quelltext, der beim Assemblieren in den eigenen Quelltext eingefügt wird. Hier sind nun alle Unterprogramme mit einem Label (Name) sowie ein Makro zum Aufruf definiert. Schauen Sie doch einfach mal in ein Include-File hinein, wenn Sie der detaillierte Aufbau interessiert. Für uns ist momentan nur wichtig, daß wir auf diese Weise ein Unterprogramm einfach mit der Sequenz

```
CALL<Libraryname> <Funktionsname>
```

aufrufen können, ohne die Adresse der Tabelle und des Offsets zu kennen. In dem Handbuch zum DEVPAC-Assembler sind auf Seite 109 die benötigten Filenamen der einzelnen Librarys aufgelistet, wobei fatalerweise aber vergessen wurde, das Oberdirectory »INCLUDE« anzugeben. So würde der Versuch scheitern, die DOS-Library mit dem Befehl

```
INCLUDE "libraries/dos_lib.i"
```

zu laden, wenn nicht zufälligerweise das aktuelle Directory »INCLUDE« ist. Narrensicher ist hingegen der Befehl

```
INCLUDE "df0:include/libraries/dos_lib.i"
```

da er unabhängig vom aktuellen Directory den Pfadnamen beim Laufwerk beginnt. Setzen Sie also bitte in Ihren eigenen Programmen immer den Vorsatz »df0:include« vor den im DEVPAC-Handbuch angegebenen Filenamen. Die Namen aller in den Include-Files definierten Funktionen finden Sie im Anhang zusammen mit den Tabellenoffsets aufgelistet.

1.4.1 Öffnen und Schließen einer Library

Wie gesagt, beginnt die Tabelle der EXEC-Library als einzige an einem festen Platz, dies ist die Adresse $0004. Die Library ist permanent geöffnet, was allein schon deshalb erforderlich ist, um die Funktion zum Öffnen einer Library benutzen zu können. Um eine Library-Funktion aufrufen zu können, muß der Basispointer der Tabelle in einer Variable mit einem bestimmten Namen abgelegt sein, damit die Include-Files darauf zugreifen können. Im Fall der EXEC-Library ist der Pointer jedoch schon innerhalb des Include-Files definiert, so daß man die Funktion einfach mit

```
CALLEXEC OpenLibrary
```

aufrufen kann. Vorher muß man jedoch einen Zeiger auf den Namen der zu öffnenden Library im Register a1 übergeben, den Sie irgendwo im Speicher ablegen müssen, z.B. »dos.library«, wenn Sie die DOS-Library öffnen möchten. Im Register d0 muß die Versionsnummer der Library übergeben werden, hier wird man im allgemeinen eine Null (Version egal) übermitteln. Die OpenLibrary-Funktion (Offset: -$0228, interessiert uns nicht) gibt nun den Pointer auf die Tabelle der geöffneten Library zurück, den wir uns für den Aufruf von Unterprogrammen merken müssen. Der Labelname muß dabei mit dem »Basepointer« (ebenfalls Seite 109 des DEVPAC-Handbuches) übereinstimmen, damit die Include-Files den Pointer benutzen können. Als Beispiel wollen wir nun einmal die MathFFP-Library öffnen:

```
move.l #ffpname,a1       ;Zeiger auf Libraryname
move.l #0,d0             ;Version egal
CALLEXEC OpenLibray      ;öffnen
move.l d0,_MathBase      ;Pointer merken

_MathBase   dc.l 0                    ;Basispointer
ffpname     dc.b "mathffp.library",0  ;Libraryname
```

Was ich Ihnen bislang verschwiegen habe, ist die Tatsache, daß das Öffnen einer Library auch schiefgehen kann. So stehen mehrere Libraries, die selten benötigt werden, nicht im ROM, sondern auf der Diskette im Unterdirectory »libs«. Sehen Sie ruhig einmal auf Ihrer Workbench-Diskette nach! Falls Sie die falsche Diskette eingelegt haben, kann die Library natürlich nicht geöffnet werden, es wird daraufhin der Wert Null zurückgegeben, was das Scheitern signalisiert.

Da der Amiga ein Multitasking-System aufweist, können natürlich alle aktiven Tasks auf eine Library zugreifen. Der Library-Manager, der Teil des Betriebssystems, der die Verwaltung der Libraries übernimmt, weiß, ob eine Library schon geladen und geöffnet wurde und wird bei weiteren Öffnungsversuchen anderer Tasks nur noch den Basispointer der Tabelle zurückgeben. Auch für das Schließen einer Library stellt uns EXEC ein Unterprogramm zur Verfügung. Dazu wird im Register a1 der Basispointer der Tabelle übergeben. Falls wir die zuvor geöffnete MathFFP-Library wieder schließen wollten, müßten wir die Befehle

```
move.l _MathBase,a1     ;Pointer holen
CALLEXEC CloseLibrary   ;Library schließen
```

einsetzen. Nach dem Schließen dürfen keine Funktionen mehr aus dieser Library aufgerufen werden. Ein vorzeitiges Schließen führt, genauso wie der Versuch, die EXEC-Library zu öffnen oder zu schließen, zu einer der beliebten Guru-Meditationen.

1.4.2 Korrekturen und Ergänzungen der Include-Files

Sehr erstaunt war ich, als ich einen katastrophalen Fehler in dem zu der Diskfont-Library gehörenden Include-File feststellen mußte. Zumindest bis zur Seriennummer 865 600 585 fehlt nämlich eine ENDM-Befehl am Schluß des Files. Dadurch werden alle weiteren Programmzeilen Ihres eigenen Quelltextes als Bestandteil des Include-Makros aufgefaßt. Unter Umständen können Sie einen Fehler in Ihrem Programm suchen, bis Sie schwarz werden, denn wer denkt denn daran, daß die gekaufte Software unkorrekt ist?

Des weiteren fehlen in dem Include-File zur Graphics-Library die Definitionen für die wichtigen Funktionen DrawEllipse und AreaEllipse. Leider bietet die Graphics-Library an sich keine Funktionen an, mit denen man die Muster von Linien und Flächen verändern könnte. Da dies theoretisch aber möglich ist, habe ich den Anlaß der Korrektur dazu genutzt, dafür drei Makros zu definieren, die Sie in Ihren eigenen Programmen unter folgenden Namen aufrufen können:

a) SetOPen
b) SetDrPt
c) SetDrMd

Genauere Angaben hierzu finden Sie in dem Kapitel über die Grafikprogrammierung. Schließlich habe ich noch eine veränderte Startup-Sequenz abgespeichert. Sie wurde aus dem Amiga-Assembler-Buch übernommen und ist wesentlich kürzer als die Original-Routine und die abgespeckte Version auf der DEVPAC-Diskette. Sie befindet sich unter dem Namen »Startup« wie die Libraries in dem Ordner »Include«, so daß sie mit dem Befehl

```
Include "df0:include/startup"
```

in Ihre Quelltexte eingebunden werden kann. Genaueres über die Startup-Sequenz finden Sie im Anhang 1. Um Ihnen die Sache möglichst einfach zu machen, habe ich die beiden korrigierten Include-Files auf der Diskette zum Buch abgelegt. Um sie auf Ihre Arbeitskopie der DEVPAC-Diskette zu übertragen, können Sie wie folgt vorgehen: Starten Sie die Diskette zum Buch, worauf die Workbench erscheint. Klicken Sie daraufhin das File CLI in der Schublade »Systems« an. Es wird nun ein CLI-Fenster geöffnet. Geben Sie jetzt den folgenden Befehl ein:

```
copy df0:include ram: all (Return)
```

Damit wird der Inhalt des Ordners »include« in die RAM-Disk kopiert. Wechseln Sie nun die Diskette zum Buch gegen Ihre DEVPAC-Arbeitskopie aus und tippen Sie die Zeile

```
copy ram: df0:include all (Return)
```

ein. So wird der Inhalt der RAM-Disk in das Unterdirektory »Include« übertragen. Da hierbei automatisch die fehlerhaften Files überschrieben werden, ist die Aktion hiermit beendet. Dieser Weg ist für Sie wesentlich einfacher, als wenn Sie jede Zeile mit dem GenAm-Assembler neu hätten eingeben müssen. Durch den Befehl

```
EndCli
```

wird das CLI-Fenster geschlossen, Sie befinden sich jetzt wieder in der Workbench und können die günstige Gelegenheit nutzen, die Beispielprogramme im Ordner »Objectcodes« auszuprobieren.

1.5 Speicherplatzreservierung beim Amiga

Im Gegensatz zu den meisten bekannten Computern verfügt der Amiga über eine dynamische Speicherverwaltung, d.h., daß sich die zu ladenden Programme, der Bildschirmspeicher etc. nicht in einem fest vorgegebenen Speicherbereich befinden, sondern theoretisch bei jedem Laden einen anderen Bereich zugewiesen bekommen können. Durch diese Art der Verwaltung können, wie es bei einem Multitasking-Betriebssystem erforderlich ist, immer mehrere Programme gleichzeitig im Speicher gehalten werden, da kein Programm auf einen bestimmten Speicherbereich angewiesen ist, wie es z.B. die C64-Umsteiger gewohnt sind.

Dem Betriebssytem wird lediglich mitgeteilt, daß Speicher benötigt wird, worauf dieser zur Verfügung gestellt wird, sofern noch welcher frei ist. Dabei interessiert nicht, welcher Task den Speicher benutzen möchte, es wird lediglich vermerkt, daß er nicht mehr für andere Tasks zur Verfügung steht, indem er aus der Liste des verfügbaren Speichers gestrichen wird.

Es wäre natürlich geradezu verheerend, auf gut Glück irgendwo seine Daten abzulegen, ohne dies dem System mitzuteilen. Deshalb existiert eine Funktion in der Exec-Library, die in 8-Byte-Schritten Speicherbelegungen vornimmt, die AllocMem-Funktion. Falls die gewünschte Speichergröße nicht durch 8 teilbar ist, wird freundlicherweise auf- und nicht abgerundet. Neben der gewünschten Speichergröße in Bytes können Sie der Funktion noch bestimmte Sonderwünsche für den Speicherbereich mitteilen. Dafür wird ein Langwort hergenommen, in dem jedes Bit eine bestimmte Eigenschaft repräsentiert.

Das Chip-Memory stellt den unteren 512-Kbyte-Bereich dar, in dem alle Daten für Grafik, Sound etc. untergebracht werden müssen, da nur hier die entsprechenden Chips Zugang haben. Mit diesem Speicherraum sollte man sorgsam umgehen, da hier auch die Grafik-Bitplanes untergebracht werden müssen, die sehr viel Speicherraum verschlingen können

Name Bedingung	Bit	Wert	Funktion
MEMF_CHIP	1	$02	Speicher muß Chip-Memory sein
MEMF_FAST	2	$04	Speicher muß Fast-Memory sein
MEMF_PUBLIC	0	$01	Speicher darf nicht verschoben werden
MEMF_CLEAR	16	$10000	Speicher wird mit Nullen gelöscht
MEMF_LARGEST	17	$20000	Speicher aus größtem Speicherblock

(siehe Kapitel 2). Die spezielle Reservierung vom Fast-Memory, d.h. dem Bereich außerhalb der unteren 512 Kbyte, ist an sich überflüssig, da bei fehlender Angabe des Speicherbereiches (Bit 1 und 2 gelöscht) zunächst versucht wird, Fast-Memory zu reservieren. Bei Amiga-500-Geräten ohne Speichererweiterung muß der Reservierungsversuch dort sowieso scheitern. Durch das MEMF_PUBLIC-Bit können Sie festlegen, daß der reservierte Speicherbereich nicht verschoben werden darf, durch das MEMF-CLEAR-Bit wird er automatisch mit Nullen gefüllt. Falls mehrere Bedingungen gleichzeitig erfüllt sein sollen, müssen die Werte durch die logische Oder-Funktion verbunden werden, d.h., Sie müssen die Werte der einzelnen Bits einfach addieren. Nach dem Aufruf der AllocMem-Funktion wird im Register D0 die Startadresse des reservierten Speicherraumes zurückgegeben, soweit der Versuch erfolgreich war. Sonst kommt eine Null zum Zeichen des Mißerfolgs zurück:

```
move.l #größe,d0        ;gewünschte Speichergröße in Bytes
move.l #bits,d1         ;Bedingungscodes
CALLEXEC AllocMem       ;Speicher reservieren
move.l d0,start         ;Startadresse merken

start  dc.l 0
```

Falls der Speicher nicht mehr benötigt wird, sollte man dies dem System mitteilen, da er sonst brachliegt und nicht mehr verwendet werden kann. Falls Sie eine 8-Megabyte-Erweiterung in Ihren Amiga eingebaut haben, ist dieses Problem wahrscheinlich von untergeordneter Bedeutung. Für die anderen Leser sei gesagt, daß hierfür eine weitere Funktion mit dem Namen FreeMem zur Verfügung steht. Ihr wird der Start des abzumeldenden Bereiches sowie erneut die Speichergröße übergeben:

```
move.l start,a1         ;Startadresse (von AllocMem erfahren)
move.l #größe,d0        ;Speichergröße in Bytes
CALLEXEC FreeMem        ;Speicher freigeben
```

Falls die Größenangabe nicht mit der bei der Reservierung übereinstimmt, brauchen Sie keine Angst zu haben, soweit die Größe beim Abmelden nicht größer als beim Reservieren ist. Im anderen Fall müssen Sie aber leider damit rechnen, daß unter Umständen für das Betriebssystem lebenswichtige Bereiche mit freigegeben wurden und irgendwann überschrieben werden. Dann meldet sich natürlich sofort der Guru zu Wort, genauso übrigens, wenn Sie einen Speicherbereich mehrmals freigeben!

Die Programmierung von Intuition

Kapitel 2

Ein wichtiger Bestandteil des Computers ist sein Betriebssystem - beim Amiga das Amiga-DOS. Die Aufgabe eines Betriebssystems besteht darin, den Computer durch den Menschen bedienbar zu machen. Im einzelnen gehören z.B. Tastaturabfrage, Bildschirmausgabe von Zeichen sowie das Diskettenhandling zu den Aufgaben.

Man kann grob gesagt zwischen zwei Arten von Betriebssystemen unterscheiden: Während die eine Gruppe die Eingabe von Befehlen über die Tastatur erwartet (MS-DOS, C64-DOS), stellt die zweite Gruppe eine sogenannte grafische Benutzeroberfläche zur Verfügung. Der wohl bekannteste Vertreter dieser Gattung ist GEM.

Bei einem solchen Betriebssystem werden die Befehle in Menüs angeboten und durch Maussteuerung ausgeführt. Der Vorteil dieser Methode besteht darin, daß der Anwender keine Befehlssequenzen lernen muß, um seinen Rechner zu bedienen, sondern sofort in die Materie einsteigen kann. Programme werden auf diese Weise wesentlich bedienerfreundlicher, da man im allgemeinen sogar ohne Handbuch mit ihnen arbeiten kann. Der Nachteil von GEM liegt in der Programmentwicklung, da an den Programmierer wesentlich höhere Ansprüche gestellt werden als bei einem konventionellen Betriebssystem. Neben der Kreation des eigentlichen Programmes muß er sich nämlich zwangsläufig auch mit der Verwaltung der Grafik und der Maussteuerung befassen. Ich möchte dies an einem Beispiel erläutern: Stellen Sie sich vor, liebe Leser, Sie wollen ein einfaches Menü mit diversen Einträgen anbieten. Welche Aktionen sind hierfür erforderlich? Nun, zunächst müssen Sie den Bildschirmausschnitt sichern, der von Ihrem Menü überdeckt wird. Anschließend ist das Menü aufzubauen. Nun müssen Sie ständig die Mausposition abfragen, um die entsprechenden Menüpunkte hervorheben (z.B. invertieren) zu können, auf denen der Mauszeiger sich befindet. Wenn schließlich ein Menüpunkt angewählt wurde, muß der urspüngliche Bildschirmausschnitt wiederhergestellt werden. Alle beschriebenen Vorgänge sind unangenehm zu programmieren, da man jeweils auf viele einzelne Bits zugreifen muß (Grafikpunkte sind ja nichts anderes als gesetzte Bits in einem speziellen Speicherbereich). Dieses einfache Beispiel erfordert schon recht großen Programmieraufwand - was soll da erst bei der Vergrößerung ganzer Windows kommen?

Nun, es gibt ein Betriebssystem, bei dem überhaupt nichts kommt. Dies heißt Amiga-DOS und umfaßt u.a. einen Teil, der Intuition heißt. Intuition hat im Gegensatz zu GEM die überaus erfreuliche Eigenschaft, daß es uns Programmierern die lästigen Nebenarbeiten

wie Verwaltung der Grafik, Aufbau von Windows, etc. praktisch komplett abnimmt. Wir bekommen nur über eine Schnittstelle (den sogenannten Message-Port) die Mitteilung, wann etwas für unser Programm Relevantes geschehen ist, z.B. das Schließ-Symbol eines Fensters angeklickt oder ein Menüpunkt ausgewählt wurde. Eine Arbeit kann uns Intuition jedoch nicht abnehmen: Wir müssen irgendwo mitteilen, welche Menüs mit wie vielen Punkten wir aufbauen möchten, was für Windows wir gerne hätten etc. Diese Informationen werden bei Intuition in umfangreichen Strukturen zur Verfügung gestellt, deren Programmierung normalerweise eine reine Fleißarbeit ist, mit Hilfe der Makros jedoch zum Kinderspiel wird.

2.1 Die Screens

Eine wesentliche Eigenschaft des Amiga besteht darin, daß er in der Lage ist, mit mehreren Bildschirmen gleichzeitig zu arbeiten. Damit ist natürlich nicht Ihr Monitor oder Fernseher gemeint, vielmehr die Grundlage für alle Ausgaben, Windows, etc. Theoretisch kann jedes Programm seinen eigenen Bildschirm benutzen, einen kennen Sie alle: den Workbench-Screen. Wenn Sie die Titelleiste des Screens anklicken, können Sie den kompletten Schirm nach unten ziehen. Hinter ihm folgt gähnende Leere. Wenn Sie jedoch mehrere Bildschirme gleichzeitig geöffnet hätten, würden Sie nun den nächsten Schirm sehen, den Sie dann auch wieder nach unten ziehen könnten. Die Screens sind also übereinandergestapelt.

Durch die Symbole rechts an der Titelleiste können Sie einen Screen ganz nach oben oder unten auf diesen Stapel legen. Eine wesentliche Eigenschaft der Screens besteht darin, daß man Sie nie nebeneinander, jedoch untereinander legen kann. Jawohl, es ist tatsächlich möglich, zwei Screens mit völlig unterschiedlichen Eigenschaften (dazu später) gleichzeitig auf einem Monitor zu betrachten. Wichtig ist hierbei nur, daß ein Screen in der linken oberen Ecke beginnen muß sowie, daß zwischen zwei Screens eine Leerzeile verbleiben muß. Man sollte sich wirklich angewöhnen, jedem Programm seinen separaten Screen zu spendieren, damit alle anderen Bildschirme von den Aktionen des Programms unbeeinflußt bleiben können. In unseren Beispielprogrammen werden wir sogar in einem Programm mehrere Screens benutzen, um die unterschiedlichen Auflösungen ausnutzen zu können. Um einen Screen zu öffnen, ruft man einfach die OpenScreen-Funktion in der Intuition-Library auf:

```
move.l #newscreen,a0    ;Zeiger auf New-Screen-Struktur
CALLINT OpenScreen      ;Funktion aufrufen
move.l d0,screenpointer ;Pointer sichern
.
.
screenpointer    dc.l 0
```

Ganz wichtig ist die Sicherung des Screenpointers. Man benötigt ihn nämlich, um z.B. Windows zu öffnen - oder auch um den Screen wieder zu schließen.

2.1.1 Die NewScreen-Struktur

Was hat es aber mit der NewScreen-Struktur auf sich? Nun, in ihr sind alle Parameter vermerkt, die wir Intuition zur Gestaltung des Screens übergeben müssen. Bevor wir in die Einzelheiten kommen, möchte ich Ihnen ein Makro für diese Struktur vorstellen, das Sie jederzeit in Ihre eigenen Programme integrieren können:

```
* Struktur für Screens

SCREEN      macro
            dc.w      0          ;linke Ecke
            dc.w      0          ;obere Ecke
            dc.w      \1         ;Breite
            dc.w      \2         ;Höhe
            dc.w      2          ;Tiefe (Bitplanes)
            dc.b      2,3        ;Farben
            dc.w      \3         ;Modus
            dc.w      15         ;Screentyp
            dc.l      0          ;Zeichensatz
            dc.l      \4         ;Screentitel
            dc.l      0          ;Gadgets
            dc.l      0          ;Bitmap
            endm
```

Die ersten beiden Wörter legen die linke obere Ecke fest. In der Regel ist dies der Punkt mit den Koordinaten (0,0). Nun zu Breite und Höhe. Diese hängen unmittelbar von dem gewählten Modus (übernächster Eintrag) ab, weshalb Sie mich bitte diesen Punkt vorziehen lassen. Die normale Auflösung unseres PAL-Amiga beträgt 320 * 256 Punkte. Daneben gibt es jedoch noch zwei weitere Auflösungen:

HIRES: Erhöht die Auflösung in X-Richtung auf 640 Punkte.

LACE: Erhöht die Auflösung in Y-Richtung auf 512 Punkte. Dieses wird durch das sogenannte Interlace-Verfahren erreicht. Dabei geschieht folgendes: Normalerweise wird das Monitorbild 50* in einer Sekunde aufgebaut, um ein möglichst flimmerfreies Bild zu erhalten. Im Zeilensprungverfahren wird jedoch nur jeweils jede zweite Zeile abgetastet, so daß sich die Auflösung verdoppelt. Da auf diese Weise aber auch die Abtastfrequenz auf 25 Bilder/s reduziert wird, fängt das Bild an zu flimmern. Deshalb habe ich diesen Modus auch noch nie benutzt und ignoriere ihn auch in diesem Buch.

Es ist logisch, daß man die Breite und Höhe des Screens nur innerhalb des gewählten Modus festlegen kann; so ist es natürlich nicht möglich, eine Höhenangabe von 512 Punkten zu tätigen, wenn man sich nicht im Interlace-Modus befindet.

Nun zu der Tiefe und den Farben des Screens, zuerst ein kleiner Überblick über die Farben des Amiga.

Jeder Grafikpunkt setzt sich aus den Grundfarben Rot, Grün und Blau zusammen. Hierfür sind die 32 Farbregister zuständig. Jedes dieser Register kann nun mit einem Wert geladen

werden, der die einzelnen Anteile der drei Grundfarben wiedergibt, und zwar jeweils von 0 bis 16. Daher kann man 16*16*16 = 4096 verschiedene Farben darstellen, jedoch bis auf zwei Ausnahmen (siehe unten) natürlich immer nur 32 gleichzeitig, da wir nur 32 Farbregister besitzen. Ich möchte an dieser Stelle noch einmal allen C64-Umsteigern den wesentlichen Unterschied darlegen: Während beim C64 jede Farbnummer eine feste, nicht veränderbare Farbe darstellt (z.B. 0 = Schwarz, 1 = Weiß, etc.), wird beim Amiga nur ein Register angesprochen, dessen Inhalt vom Benutzer frei wählbar ist. Um einem Grafikpunkt eine bestimmte Farbe zuzuordnen, wird die Nummer des Farbregisters in Bits zerlegt: Man benötigt für 32 Farben 5 Bit. Der Haken an der Sache ist jedoch, daß ein Grafikpunkt nur durch ein Bit repräsentiert ist: entweder der Punkt (und das Bit) sind gesetzt oder nicht. Um fünf Bit pro Punkt zu erhalten, muß man also fünf Grafikspeicher (Bitplanes) zu Hilfe nehmen. Dabei wird pro Bitplane jedem Grafikpunkt genau ein Bit zugeordnet: In der Bitplane 0 jeweils das nullte Bit, in der Bitplane 1 das jeweils erste, usw. (Bild 2.1). Jede Bitplane benötigt jedoch Speicherplatz. Daher kann man wahlweise auch weniger Farben benutzen und damit Bitplanes einsparen, je nachdem, wie viele Bits benötigt werden: Möchte man z.B. nur eine Bitplane benutzen, kann man nur zwei Farben darstellen, mit drei Bitplanes immerhin schon deren acht. In dem obigen Makro habe ich zwei Bitplanes gewählt, mit denen wir vier Farben darstellen können. Eine Einschränkung des HiRes-Modus besteht darin, daß man nur mit vier Bitplanes und mit maximal 16 Farben arbeiten kann. Beim Amiga wird jedoch erhöhte Farbenvielfalt nur mit zusätzlichem Speicherplatzbedarf bezahlt, nicht mit einer halbierten Grafikauflösung wie beim C64.

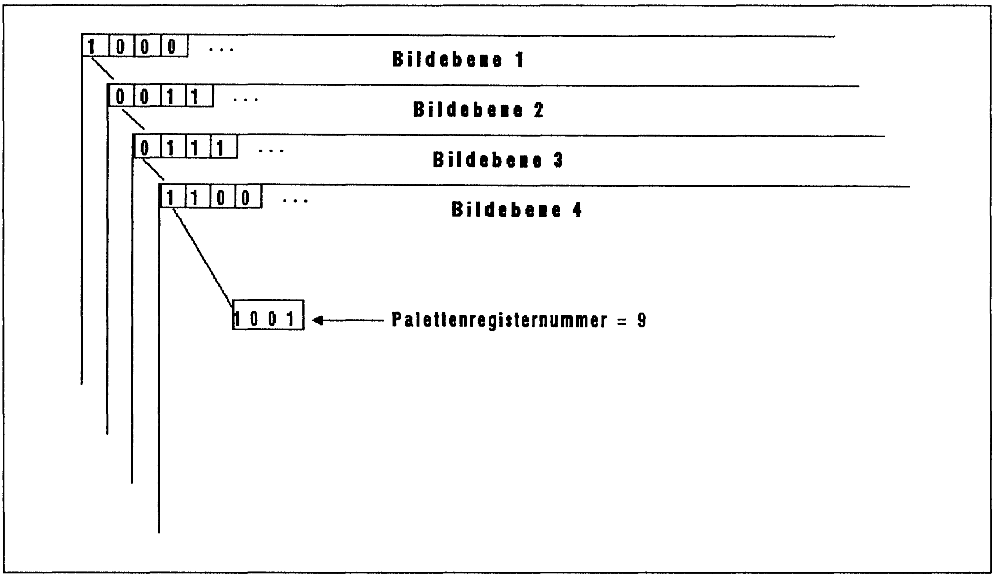


Bild 2.1: Bildschirmebenen und Farbkodierung

Die Farbregisternummern, die vor der Tiefe des Screens angegeben werden müssen, stellen Zeichen- und Hintergrundfarbe dar. Man kann sie natürlich nur aus den Werten auswählen, die durch die Tiefe ermöglicht sind. So kann man natürlich nicht nur mit zwei Bitplanes arbeiten und dann eine Nummer größer als 3 angeben. In der Voreinstellung enthalten die einzelnen Register folgende Farben:

	0: Blau 1: Weiß
bei 2 Bitplanes:	2: Schwarz 3: Rot
bei 3 Bitplanes:	4: Blau 5: Violett 6: Türkis 7: Weiß
bei 4 Bitplanes:	8: Schwarz 9: Rot 10: Grün 11: Braun 12: Blau 13: Blau 14: Grün 15: Grau

Neben den Modi LORES und HIRES gibt es noch zwei weitere, die es erlauben, mit wesentlich mehr Farben gleichzeitig zu arbeiten:

HAM: Der »Hold and Modify«-Modus erlaubt es, alle 4096 Farben gleichzeitig darzustellen. Dafür sind jedoch sechs Bitplanes erforderlich. Dabei wird immer nur eine der drei Grundfarben Rot, Grün und Blau geändert, während die anderen beiden von dem letzten Punkt übernommen werden. Die Bitplanes 4 und 5 geben hierbei an, was mit den Bits der Ebenen 0–3 zu geschehen hat:

Ebene		Aufgabe der Ebenen 0–3
4	5	
0	0	Die Bitkombination der Ebenen 0–3 gibt an, aus welchem Farbregister der Punkt die Farbe beziehen soll (0–15).
0	1	Die Bitkombination der Ebenen 0–3 gibt den neuen Wert der Grundfarbe Blau an. Die Werte der Farben Rot und Grün werden vom Vorpunkt übernommen.
1	0	Die Bitkombination der Ebenen 0–3 gibt den neuen Wert der Grundfarbe Rot an. Die Werte der Farben Blau und Grün werden vom Vorpunkt übernommen.
1	1	Die Bitkombination der Ebenen 0–3 gibt den neuen Wert der Grundfarbe Grün an. Die Werte der Farben Rot und Blau werden vom Vorpunkt übernommen.

Man kann den HAM-Modus auch mit nur vier Bitplanes betreiben, wobei die fehlenden Bits der Ebene 5 als 0 gesetzt werden. Man kann also wahlweise die normale Farbgebung oder die Modifikation des Rot-Anteils wählen. Im übrigen ist er nicht bei gleichzeitigem INTERLACE- und/odder HIRES-Modus zu verwenden.

EXTRA HALFBRIGHT: Dieser Modus läßt die Darstellung von 64 Farben gleichzeitig zu. Dabei wird eine sechste Bitplane benötigt, durch die sich 32 weitere Farbregister mit den Nummern 32–63 ansprechen lassen. Nun können Sie natürlich nicht 32 völlig neue Farben definieren, vielmehr werden Ihnen die Werte der Register 0–31 mit jeweils halbierter Helligkeit zugeordnet. Da eine neue Farbe jedoch nur dann entsteht, wenn wenigstens eine der drei Grundfarben gleich bleibt, nenne ich diesen Modus auch immer den »Schummelmodus«.

Sie können sich vorstellen, daß die Screens je nach gewähltem Modus einen unterschiedlichen Platzbedarf aufweisen. Man kann ihn nach folgender Formel berechnen:

Bytes = (Breite<Pixel>*Höhe<Pixel>*Bitplanes)\8

Da jeder Punkt ein Bit in jeder Bitplane benötigt und jeweils 8 Bit ein Byte bilden, ist sie wohl leicht einzusehen. Hier nun eine Tabelle mit dem Platzbedarf aller Modi.

Modus	Breite*Höhe	Tiefe	Farben	Speicherbedarf
LORES	320*256	1–5	2–32	10 240– 51 200 Byte
LORES + INTERLACE	320*512	1–5	2–32	20 480–102 400 Byte
HIRES	640*256	1–4	2–16	20 480– 81 920 Byte
HIRES + INTERLACE	640*512	1–4	2–16	40 960–163 840 Byte
HAM	320*256	5–6	4096	51 200– 61 440 Byte
EXTRA HALFBRIGHT	320*256	6	64	61 440 Byte
EXTRA HALF.+INTERLACE	320*512	6	64	122 880 Byte

Sie sehen, liebe Leser, daß das Arbeiten mit mehreren Screens durchaus zu Speicherproblemen führen kann, da sich alle Screen-Daten in den unteren 512 Kbyte (Chip-RAM) des Amiga befinden müssen. Nur dort nämlich hat der Grafik-Chip Zugang zu diesen Daten. Nun jedoch zurück zu unserem Makro: Das nächste Wort beinhaltet den Modus, den wir eben so ausführlich besprochen haben. Dazu kommen noch 2 Bit für die Verwendung von Sprites und einer externen Videoquelle:

Bit	Wert	Name	Bedeutung
1	2	GENLOCK VIDEO	Erlaubt Verwendung einer externen Video-Quelle über ein Interface. Das Videobild erscheint unter dem Amiga-Bild.
2	4	INTERLACE	Verdoppelt Zeilenanzahl (s.o.)
7	128	EXTRA HALFBRIGHT	Darstellung von 64 Farben (s.o.)
11	2048	HAM	Darstellung von 4096 Farben (s.o.)
13	8192	VP HIDE	Screen wird erzeugt, aber nicht dargestellt.
14	16384	SPRITES	Erlaubt Verwendung von Sprites
15	32768	HIRES	Verdoppelt Spaltenzahl (s.o.)

Das folgende Wort ist für den Screen-Typ zuständig. Hier sollten Sie immer den Wert 15 eintragen, was bedeutet, daß Sie einen Benutzer-Screen öffnen möchten. Theoretisch könnten Sie auch eine Workbench öffnen. Wenn man sich hier jedoch nicht gut auskennt, ist der Guru nicht mehr weit.

Das nächste Langwort muß einen Zeiger auf den zu verwendenden Zeichensatz beinhalten. Da man ohne größere Umstände nur auf den Standard-ROM-Zeichensatz zugreifen kann, trägt man eine Null ein. (Natürlich werden wir im Laufe des Buches auch auf die Disk-Zeichensätze zu sprechen kommen, nur Geduld!) Nun folgt ein Zeiger auf den Titel des Screens. Das nächste Langwort enthält einen Zeiger auf eine sogenannte Gadget-Struktur. Neben den Ihnen bekannten Gadgets von der Workbench (Close-Gadget, etc.) ist es nämlich möglich, eigene Gadgets zu implementieren. Da dies jedoch im allgemeinen nur in Windows sinnvoll ist, geben wir hier eine Null (keine Gadgets) ein. Ebenfalls eine Null erhält das letzte Langwort. Dies bedeutet, daß wir keine eigene Bitmap verwenden, was auch nicht erforderlich ist, da diese automatisch von Intuition beim Öffnen des Screens angelegt wird. Ein Makroaufruf könnte daher wie folgt lauten:

```
SCREEN 640,256,$C000,titel
titel dc.b "Mein erster Screenr",0
```

Hier würde ein Screen im HIRES-Modus eröffnet, der die Verwendung von Hardware-Sprites zuläßt (Bits 14 und 15 im Modus-Flag gesetzt). Natürlich können Sie noch weitere Angaben wie z.B. die Farben als Parameter übergeben, da der DEVPAC ja 9 Parameter pro Makro erlaubt. Ich komme jedoch mit diesen vier Parametern gut aus.

2.1.2 Die Screen-Struktur

Nun jedoch zu einer anderen Frage: Was für ein Langwort bekommen wir eigentlich zurück, wenn wir den Screen öffnen? Nun, dies ist kein Fantasiewert, der nur zum Schließen des Screens benötigt wird, sondern ein Zeiger auf die Screen-Struktur. Diese darf nicht mit der NewScreen-Struktur unseres Makros verwechselt werden, sie wird nämlich von Intuition in der OpenScreen-Funktion erzeugt. Die Screen-Struktur enthält nun wesentlich mehr Daten über den Screen, die jedoch sehr wichtig für den Programierer sein können:

Offset	Typ	Name	Funktion
$000	L	Next Screen	Zeiger auf nächsten Screen
$004	L	First Window	Zeiger auf erste Fenster-Struktur
$008	W	LeftEdge	Linke Ecke (x-Koordinate)
$00A	W	TopEdge	Obere Ecke (y-Koordinate)
$00C	W	Widht	Breite in Pixeln
$00E	W	Height	Höhe in Pixeln
$010	W	MouseY	Mausposition (y-Koordinate)
$012	W	MouseX	Mausposition (x-Koordinate)
$014	W	Flags	Screen-Flags (wie in NewScreen)
$016	L	Title	Zeiger auf Screen-Titel
$01A	L	DefaultTitle	Z. a. Window-Titel für Windows ohne Screen-Titel
$01E	B	BarHeight	Dimension der Titelleiste des
$01F	B	BarVBorder	Screens
$020	B	BarHBorder	
$021	B	MenuVBorder	Dimension der Menüleiste
$022	B	MenuHBorder	
$023	B	WBorTop	Dimension der Titelleiste

Offset	Typ	Name	Funktion
\$024	B	WBorLeft	Aller Windows
\$025	B	WBorRight	Screens
\$026	B	WBorBottom	
\$027	B	Adjusttoword	Adresse begradigen
\$028	L	Font	Zeiger auf Zeichensatz
\$02C-\$053		ViewPort	View-Port-Struktur des Screens. Diese enthält weitere x- und y-Dimensionen sowie Pointer zu einer Liste, die die Videodarstellung steuert (Chopper).
\$054-\$0B7		RastPort	Rast-Port-Struktur des Screens. Diese enthält Angaben, die für Grafik benötigt werden, wie Liste über die verwendeten Zeichenstifte, Füllmuster.
\$0B8-\$0DF		BitMap	Externe Bitmap-Struktur
\$0E0-\$145		LayerInfo	Layer-Info-Struktur (relativ unwichtig)
\$146	L	FirstGadget	Zeiger auf erste Gadget-Struktur
\$014A	B	DetailPen	Zeichenfarbe
\$014B	B	BlockPen	Hintergrundfarbe
\$014C	W	SaveColor	Enthält Farbwert, der bei DisplayBeep (s.u.) verwendet wird.
\$014E	L	BarLayer	Zeiger auf Darstellung der Titelleiste
\$0152	L	ExtData	Hier kann der Anwender Zeiger auf
\$0156	L	UserData	eigene Strukturen eintragen

Wenn Ihnen manche der Begriffe noch nichts sagen, macht dies überhaupt nichts. Im Laufe des Buches werden die meisten schrittweise eingeführt.

Das Wichtigste fehlt jedoch noch: Die Routine zum Schließen des Screens. Hierfür laden Sie einfach den Screen-Pointer in das Register A0 und rufen die Close-Screen-Routine auf:

```
move.l screenpointer,a0
CALLINT CloseScreen
```

2.1.3 Manipulation von Screens

Zum Schluß dieses Abschnitts möchte ich Ihnen noch vier weitere Routinen vorstellen, mit denen Sie den Screen manipulieren können:

`Move-Screen`

Durch die Move-Screen-Routine erreichen Sie softwaremäßig das, was Sie auch mit der Maus anstellen können: das Verschieben des Screens. Der Aufruf geschieht mit

```
move.l screenpointer,a0      ;Pointer laden
move.l #points,d1            ;Zahl der Punkte, um die in y-Richtung verschoben
                              wird clr.l d0
CALLINT MoveScreen           ;Routine aufrufen
```

`ScreenToBack`

Durch diese Funktion wird der Screen in den Hintergrund gebracht und von allen anderen Screens verdeckt. Sie erreichen dieses auch mit der Maus, imdem Sie das entsprechende Gadget in der Titelleiste anwählen.

```
move.l sreenpointer,a0
CALLINT ScreenToBack
```

`ScreenToFront`

Das Gegenstück zu b), der Screen wird nach vorne geholt.

```
move.l sreenpointer,a0
CALLINT ScreenToBack
```

`DisplayBeep`

Diese Routine läßt den Screen kurz aufblinken. Sie wird u.a. vom Basic-Interpreter dazu verwendet, Syntax-Fehler anzuzeigen. Die Blinkfarbe wird automatisch von Intuition gesetzt, Sie können sie aus der Screen-Struktur auslesen (siehe oben).

```
move.l screenpointer,a0
CALLINT DisplayBeep
```

Wenn Sie statt des Screenpointers den Wert Null laden, blinken alle Intuition-Bildschirme. Damit wären wir schon am Ende des Screen-Abschnitts angelangt. Als Fazit kann man wohl festhalten, daß der Amiga uns sehr vielfältige Möglichkeiten zur Bildschirmgestaltung mitgibt, die von uns in einer einfachen Struktur an Intuition übergeben werden. Durch einen Screen werden jedoch nur die Randbedingungen geschaffen. Wenn wir etwas auf dem Bildschirm darstellen möchten, benötigen wir ein weiteres Element: die Windows.

2.2 Die Windows

Wenn Sie schon von den Screens begeistert waren, liebe Leser, müßten Sie nach der Lektüre des nun folgenden Abschnitts normalerweise in Euphorie geraten. Denn die Windows des Amiga stellen alles in den Schatten, was bislang bezüglich grafischer Benutzeroberflächen bekannt war. Sicherlich kennen Sie schon einige herausragende Merkmale von der Workbench: Windows können verschoben, vergrößert, verkleinert und durch weitere Windows verdeckt werden, ohne daß ihr Inhalt verlorenginge. Dazu kann jedes Window sein eigenes Menü besitzen und auf die eigenartigsten Dinge wie von Geisterhand reagieren. Das beste Beispiel hierfür ist das Einlegen der Workbench-Diskette auf die Aufforderung in der linken oberen Bildschirmecke. (Leser mit nur einem Laufwerk wissen, was ich meine...)

Nun, die Programmierung eines Windows ist nicht wesentlich komplizierter als die eines Screens, da sie nach dem gleichen Muster abläuft: In einer Struktur mit dem Namen NewWindow werden die Daten für Intuition abgelegt, das Öffnen eines Windows geschieht völlig analog zum Screen:

```
move.l #newwindow,a0
CALLINT OpenWindow
move.l d0,windowpointer
.
.
windowpointer  dc.l 0
```

Auch hier bekommen wir von Intuition einen Pointer auf eine weitere Struktur zurück, wie wir es aus dem letzten Abschnitt kennen. Aus dieser Struktur sind Daten wie Mausposition, etc. ablesbar, dazu jedoch später.

2.2.1 Die NewWindow-Struktur

Zunächst möchte ich Ihnen wieder ein Makro vorstellen, das Sie modifiziert oder im Originalzustand in eigene Programme integrieren können:

```
* Struktur für Windows

WINDOW      macro
            dc.w       \1       ;linke Ecke
            dc.w       \2       ;obere Ecke
            dc.w       \3       ;Breite
            dc.w       \4       ;Höhe
            dc.b       0,1      ;Farben
            dc.l       \5       ;Flags für Events
            dc.l    $100f       ;Window Flags
            dc.l       \6       ;Zeiger auf erstes Gadget
            dc.l       0        ;Check Mark
            dc.l       \8       ;Windowname
```

```
\7          dc.l      0       ;Screenpointer
            dc.l      0       ;Bit Map
            dc.w      \3      ;Mindest-Breite
            dc.w      \4      ;Mindest-Höhe
            dc.w      \3      ;maximale Breite
            dc.w      \4      ;maximale Höhe
            dc.w      15      ;Screen-Typ
            endm
```

Die ersten beiden Parameter legen die linke, obere Ecke des Windows fest. Man muß jedoch beachten, daß diese Koordinaten relativ zu denen des Screens zu sehen sind (siehe Bild 2.2.1)! Falls also z.B. der Screen bei (x,y) = (0,30) beginnt und das Window bei (x,y) = (20,20), so würden die absoluten Startkoordinaten des Windows (x,y) = (20,50) lauten.

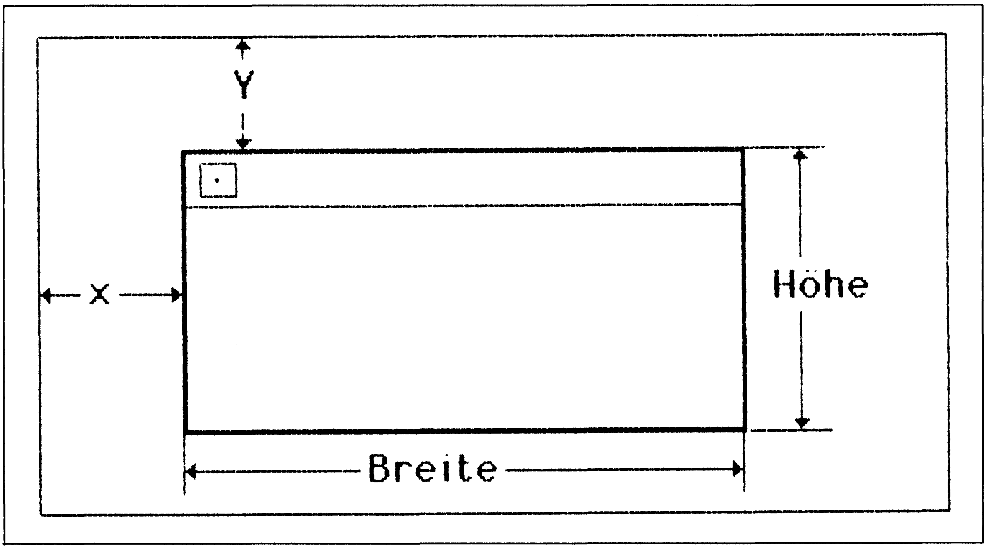


Bild 2.2: Lage und Maße eines Windows

Die beiden folgenden Wörter legen die Anfangshöhe und -breite des Windows fest. Diese sind im Gegensatz zu denen des Screens jedoch nicht bindend, da man ein Window (in der Regel) mit der Maus vergrößern und verkleinern kann. Diese Daten beziehen sich auf den relativen Ursprung des Windows (Bild 2.2).

Die Farbwahl geschieht analog zu der des Screens. Natürlich kann man auch ein Window nur in den Farben darstellen, die man durch den Screenmodus und die Anzahl der Bitplanes auch erreichen kann. Wenn ein Screen nur eine Bitplane besitzt, können auch alle Windows dieses Screens nur die beiden Farben der Register 0 und 1 benutzen. Eine

Besonderheit besitzt der normalerweise unsinnige Wert -1: mit ihm werden die Farben des Screens übernommen.

Das folgende Langwort beinhaltet die sogenannten Event-Flags. Dazu muß ich jedoch ein wenig weiter ausholen. Wie ich schon oben erwähnt habe, wird die Steuerung der Grafik und damit auch der Windows komplett von Intuition übernommen. Diese erfreuliche Tatsache hat jedoch einen Haken: Wie erfährt der Programmierer, wenn der Anwender z.B. ein Menü angewählt hat oder das Close-Gadget angeklickt hat? Die Antwort ist der sogenannte Intuition Direct Communication Message Port (IDCMP). Er stellt gewissermaßen die Verbindung zwischen Intuition und dem Programm her und meldet bestimmte Ereignisse. Im übernächsten Abschnitt werden wir dieses Thema ausführlich behandeln. Zu diesem Zeitpunkt müssen wir nur wissen, was uns Intuition denn überhaupt melden soll. Jedes Ereignis besitzt einen bestimmten Wert (Event-Flag), den wir in die NewWindow-Struktur eintragen müssen:

Bit	Wert	Name	Bedeutung: Nachricht bei
0	1	SIZEVERIFY	Anwender möchte Fenstergröße verändern
1	2	NEWSIZE	Anwender hat Window-Größe verändert.
2	4	REFRESHWINDOW	Refreshmodus (s.u.)
3	8	MOUSEBUTTONS	Drücken einer Maustaste
4	16	MOUSEMOVE	Die x-/y-Koordinaten der Maus werden gemeldet
5	32	GADGETDOWN	Anwender hat Gadget angeklickt
6	64	GADGETUP	Gadget wurde angeklickt und losgelassen
7	128	REQSET	Meldung, wenn erster Requester gewählt
8	256	MENUPICK	Anwahl eines Menüpunktes
9	512	CLOSEWINDOW	Schließen des Fensters
10	1024	RAWKEY	Tastendruck (Scan-Code)
11	2048	REQVERIFY	Meldung, bevor Requester erscheint
12	4096	REQCLEAR	Meldung, wenn letzter Requester weg
13	8192	MENUVERIFY	Meldung, bevor Menu aufgebaut wird
14	16 384	NEWPREFS	Neue Preferences eingestellt
15	32 768	DISKINSERTED	Diskette eingelegt
16	65 536	DISKREMOVED	Diskette herausgenommen

Bit	Wert	Name	Bedeutung: Nachricht bei
17	131 072	WBENCHMESSAGE	Mitteilung von Workbench
18	262 144	ACTIVEWINDOW	Meldung, wenn Window aktiviert wurde
19	524 288	INACTIVEWINDOW	Meldung, wenn Window inaktiviert wurde
20	1 048 576	DELTAMOVE	Mauskoordinaten relativ melden
21	2 097 152	VANILLAKEY	Tastendruck (ASCII-Code)
22	4 194 304	INTUITICKS	Message ca. 10x pro Sekunde ausgeben

Keine Angst, im Kapitel 2.5 werden alle 23 Flags ausführlich durchleuchtet werden! Wichtig ist jetzt nur, daß man die einzelnen Werte addieren muß, wenn man mehrere Flags setzen möchte, so wird z.B. die Anwahl eines Menüpunktes und Einlegen einer Diskette gemeldet, wenn man den Wert 32 768 + 256 = 33 024 in die NewWindow-Struktur einträgt.

Als nächstes folgt wiederum ein Langwort, in das die sogenannten Window-Flags eingetragen werden müssen. Diese dürfen nicht mit den IDCMP-Flags verwechselt werden, obwohl sie oft sehr eng zusammenhängen:

Bit	Wert	Name	Bedeutung: Nachricht bei
0	1	WINOWSIZING	Fenstergröße veränderbar
1	2	WINDOWDRAG	Fenster verschiebbar
2	4	WINDOWDEPTH	Fensterüberlagerung möglich
3	8	WINDOWCLOSE	Window schließbar

Durch diese 4 Bit werden die sogenannten Systemgadgets repräsentiert, die Sie schon von der Workbench kennen (Bild 2.3). Es ist sicherlich kein Fehler, sie in jedem Window zu installieren. Deshalb besitzt unser Makro auch den Wert $0F (15=1+2+4+8). Wo der Rest (die $1000) herkommt, werden wir nun klären.

Die folgenden beiden Bits beziehen sich auf das WINDOWSIZING-Gadget.

Bit	Wert	Name	Bedeutung: Nachricht bei
4	16	SIZEBRIGHT	Gagdet wird an der rechten Seite angebracht.
5	32	SIZEBBOTTOM	Gadget wird an der unteren Seite angebracht.

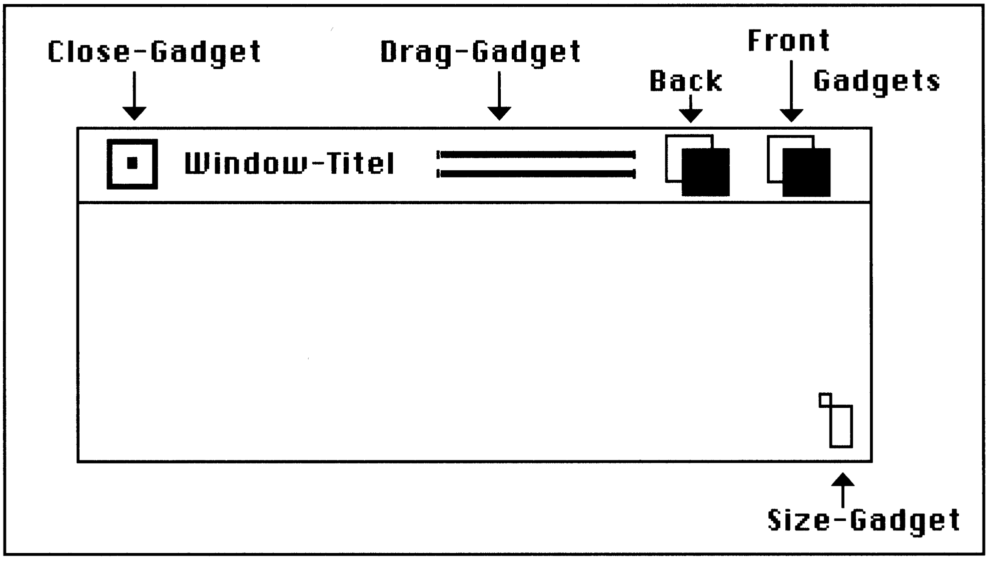


Bild 2.3: Die System-Gadgets eines Windows

Nun folgen 2 Bit, durch die der sogenannte Refreshmodus bestimmt wird:

Bit	Wert	Name	Bedeutung: Nachricht bei
6	64	SIMPLE_REFRESH	
7	128	SUPER_BITMAP	

Der Amiga kennt neben diesen beiden noch einen dritten Modus, den sogenannten SMART_REFRESH. Er ist aktiviert, wenn keiner der anderen beiden ausgewählt wurde. Nun aber zur Bedeutung des Refreshing.

Dies bedeutet nichts anderes als Fenstererneuerung (Bild 2.4). Sie ist immer dann erforderlich, wenn ein Fenster (teilweise) durch ein weiteres Fenster verdeckt und letzteres wieder geschlossen wird. Bildlich gesprochen liegen zwar beide Fenster übereinander, da wir jedoch nur einen Bildschirm haben, werden die Daten des verdeckten Window-Teils wohl oder übel gelöscht. Auch bei der Verschiebung eines Fensters müssen gewaltige Mengen von Grafikdaten erneuert werden. Sie werden es wahrscheinlich schon geahnt haben, auch diese Mordsarbeit nimmt uns Intuition ab. Wir können jedoch festlegen, wieviel Erneuerung wir unserem Window gönnen.

Im SIMPLE_REFREH-Modus operiert Intuition nur innerhalb des Video-Rams, wodurch kein zusätzlicher Speicherbedarf resultiert. Daher darf dieser Modus nur dann angewendet werden, wenn sichergestellt ist, daß das Fenster nicht durch ein anderes verdeckt werden

kann (z.B. wenn nur ein Fenster existiert...). In diesem Fall müßten nämlich die Daten des verdeckten Teils in einen externen Speicherbereich gerettet werden, um sie nachher wieder in das Video-RAM zurückkopieren zu können. Allerdings ist es ohne weiteres möglich, ein Fenster in diesem Modus zu verschieben, da die Daten nur innerhalb des Video-RAMS kopiert werden müssen. Ein Nachteil dieses Modus besteht darin, daß bei einer eventuellen Fenstervergrößerung die neu hinzugekommenen Bildteile vom Programmierer gefüllt werden müssen.

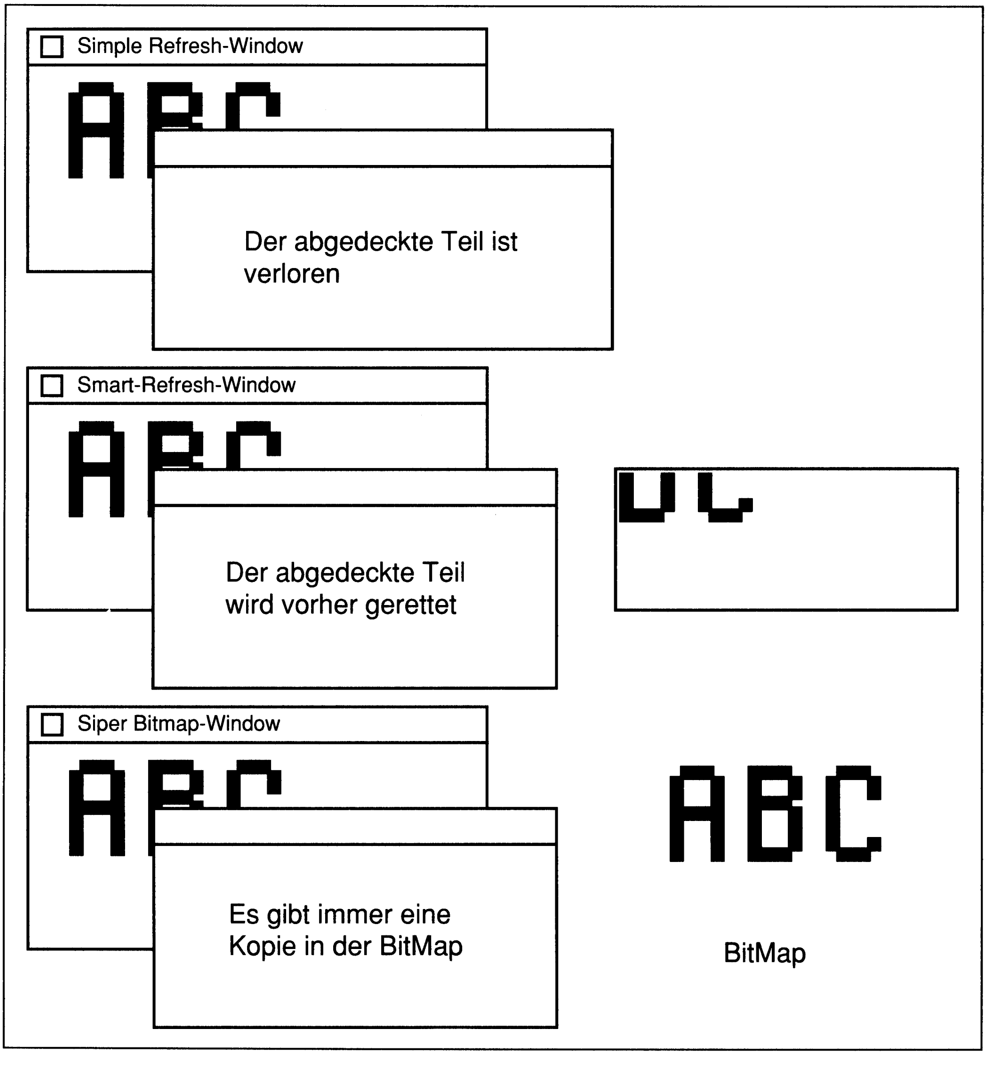


Bild 2.4: Die Refresh-Methoden im Vergleich

Der SMART_REFRESH ist genau dann einzusetzen, wenn der obige Modus versagt, nämlich dann, wenn es gilt, überdeckte Fensterteile zu restaurieren oder Fenster zu vergrößern. Ein Problem bleibt jedoch auch hier ungelöst: Wenn man das Fenster verkleinert und dann wieder vergrößert, fehlt der weggeschnittene Teil. Da nämlich keine Überlappung stattfand, wurden auch keine Daten gerettet. Man kann sich jedoch mit einem Trick helfen, indem man die Minimalhöhe und -breite des Windows (siehe unten) auf die Startwerte setzt. Damit ist es nicht möglich, das Window kleiner zu machen, als zu dem Zeitpunkt, wo es auf dem Bildschirm erschien.

Der SUPER_BITMAP-Modus schließlich ist das Nonplusultra des Refreshings. Hierbei wird nämlich zu jedem Zeitpunkt eine komplette Kopie des Windows in einem externen Puffer gehalten, wodurch Sie natürlich soviel verkleinern und vergrößern können, wie es Ihnen Spaß macht. Weniger Spaß bereitet dieser Modus jedoch dem RAM, da ja je nach Zahl der Bitplanes eine Unmenge an Speicher verbraucht werden kann (siehe 2.1). Dazu ist es notwendig, den Speicher für Intuition zu reservieren. Es sind folgende Schritte erforderlich:

a) BitMap mit InitBitMap (Graphics-Library) initialisieren
b) Mit AllocRaster (Graphics-Library) Speicher beschaffen
c) Zeiger auf BitMap-Struktur in NewWindow-Struktur eintragen

Da man jedoch bei Verwendung von SuperBitMap-Windows meistens in ernste Schwierigkeiten mit dem Speicher kommt und man mit dem Smart-Refresh Modus auch gut bedient ist, habe ich in diesem Buch auf diese Windows verzichtet. Es folgen drei Flags, die den Windowtyp festlegen:

8	256	BACKDROP
10	1024	GIMMEZEROZERO
11	2048	BORDERLESS

Das BORDERLESS-Window wird, wie der Name sagt, dadurch charakterisiert, daß der übliche Rahmen feht. Dieses scheint auf den ersten Blick sinnlos zu sein. Sie können jedoch elegant den Bildschirm löschen, indem Sie ein solches Window öffnen, welches den gesamten Bildschirm überdeckt. Der Trick an dieser Sache ist, daß keine Grafikdaten zerstört werden, sondern nur von dem leeren Window verdeckt werden. Wenn Sie nämlich das Borderless-Window wieder schließen, sind alle verdeckten Grafiken wieder sichtbar.

Das BACKDROP-Window scheint auf den ersten Blick noch überflüssiger zu sein. Die »herausragende« Eigenschaft dieses Windows besteht nämlich darin, daß es beim Öffnen automatisch in den Hintergrund gebracht wird und dort auch bleibt, egal, wie viele Fenster man mit Hilfe des Back-Gadgets versucht, hinter das Backdrop-Window zu legen. Die einzige Möglichkeit, ein solches Window nach vorne zu bringen, besteht darin, weitere Backdrop-Windows zu öffnen. Der Sinn eines solchen Windows besteht darin, den Anwender zu zwingen, seine Operationen in diesem Window durchzuführen. Da er es nicht nach vorne klicken kann, kann man als Programmierer z.B. bei Zeichenprogrammen sicher sein, daß nicht versucht werden kann, in anderen Windows zu malen.

Das GIMMEZEROZERO-Window schließlich verlegt den Nullpunkt an den unteren Rand der Titelleiste. Während Sie also bei normalen Windows theoretisch nach Herzenslust alle Gadgets sowie den Window-Titel übermalen könnten, ist dies hier nicht möglich, da der Koordinatenursprung erst unterhalb dieser Leiste beginnt. Geöffnet wird ein Gimmezero-zero-Window jedoch mit den Außenmaßen wie bei allen anderen Windows auch. Aus der Window-Struktur, die wir am Ende dieses Abschnitts behandeln werden, können Sie dann die Ausmaße des »inneren« Windows auslesen und dann mit diesen weiterrechnen. Man könnte nun denken, daß ein Gimmezerozero-Window grundsätzlich Verwendung finden sollte. Man darf jedoch einen gewaltigen Nachteil nicht übersehen: Titelleiste und inneres Fenster werden von Intuition als getrennte Speicherbereiche verwaltet. Dadurch wird erstens mehr RAM zur Verwaltung benötigt, und zweitens, was noch viel schlimmer ist, die Standardoperationen Verschieben, Verkleinern und Vergrößern laufen wesentlich langsamer ab, besonders wenn man mehrere Windows gleichzeitig geöffnet hat. Das Bit 9 sollte nicht unterschlagen werden, paßte aber nicht so recht zu den Window-Typen. Deshalb wird es schleunigst nachgeholt:

```
9     512  REPORTMOUSE
```

Das Bit muß immer dann gesetzt werden, wenn man über die aktuelle Mausposition informiert werden möchte. Kritische Leser werden nun einwenden, daß das schon mit dem vierten Bit der IDCMP-Flags geschehen ist! Richtig, nur leider ist der Haken an der Sache, daß das IDCMP-Flag so lange wirkungslos bleibt, bis nicht auch das Window-Flag Reportmouse gesetzt wurde. Der Zweck dieser Maßnahme bleibt für mich unverständlich.

Eine Variante erhalten Sie, wenn Sie das Bit 17 setzen: Hierdurch wird verhindert, daß Sie eine Refresh-Meldung erhalten. Wenn Sie das IDCMP-Bit 2 gesetzt haben, widersprechen sich offenbar beide Bits. Das traurige Ergebnis besteht darin, daß das Window- das IDCMP-Flag schlägt: Sie erhalten keine Meldung...

```
12    4096  ACTIVATE
```

Ein Setzen dieses Bits sorgt dafür, daß dieses Window nach dem Öffnen sofort aktiviert wird. Da in unseren Beispielprogrammen dies immer der Fall ist, wurde in dem obigen Makro eine $1000 (dez. 4096) eingeblendet.

In unserem Makro folgt nun ein Zeiger auf die Struktur des ersten Gadgets. Man muß dazu wissen, daß man neben den vier Systemgadets auch noch weitere eigene Gadgets installieren kann, mit denen man z.B. komfortable Eingaben machen kann (Kapitel 2.7). Jedes Gadget besitzt, wie könnte es anders sein, eine Struktur, in der Eigenschaften und Aussehen definiert werden. Wenn keine Gadgets vorhanden sind, trägt man eine Null ein.

Das folgende Langwort enthält wiederum einen Zeiger, diesmal auf eine Grafik. Wie Sie wahrscheinlich wissen, ist es möglich, einzelne Menüpunkte mit einem Haken zu versehen. Falls Ihnen der Haken nicht gefällt, können Sie eine eigene Grafik, ein sogenanntes Image (Abschnitt 2.8) definieren, durch die der Haken ersetzt werden kann. Falls Sie mit dem Haken zufrieden sind (wie unser Makro), tragen Sie auch hier eine Null ein.

Nun folgt ein Zeiger auf den Window-Titel. Wie schon beim Screen muß auch hier der Text mit einem Null-Byte abgeschlossen werden. Ein Text wird abgeschnitten, der breiter als das Window ist. Falls Sie einen eigenen Screen benutzen, müssen Sie jetzt den Pointer auf die Screen-Struktur eintragen. Bei der Benutzung eines Makros ist dieses ein kritischer Punkt, da man die Adresse des Makros innerhalb seines Programmes nicht kennt, jedoch erst nach Öffnen des Screens den Pointer eintragen kann. Daher mußte ein Label herhalten, was die entsprechende Speicherstelle markiert. Man kann dann leicht den Pointer in die NewWindow-Struktur eintragen.

Falls Sie als Refresh-Modus den SUPER_BITMAP-Modus gewählt haben, ist es erforderlich, daß Sie den Zeiger auf die BitMap in das nun folgende Langwort eintragen. Dies müßte theoretisch wiederum mit einem Label passieren, da man die BitMap-Adresse auch erst im laufenden Programm erfährt. Da wir jedoch auf ein solches Window verzichten, tragen wir eine Null ein. Es folgen zwei Wörter, die die Mindestbreite und Höhe des Windows angeben. Diese Werte können nicht unterschritten werden, selbst falls der Anwender wie ein Wilder auf seiner Maus herumtrommelt. Es kann sinnvoll sein, ein Unterschreiten der Startgröße des Windows nicht zuzulassen. Damit entgeht man nämlich beim Refresh auf jeden Fall der Überraschung, daß Windowteile weggeschnitten werden. Nur im SUPER_BITMAP-Modus kann man sich gemütlich zurücklehnen und erlauben, daß das Fenster (fast) verschwindet... Die nächsten Wörter geben analog die maximal zulässigen Maße des Windows an. Damit kann man z.B. verhindern, daß relevante Ausschnitte des Hintergrundes verdeckt werden können. Als letztes folgt wieder ein Langwort. Hier muß der Screen-Typ eingetragen werden, wie dies schon bei der NewScreen-Struktur der Fall war. Zu guter Letzt schulde ich Ihnen natürlich noch den Aufruf des Window-Makros. Hier ist er:

```
WINDOW 0,0,320,256,$300,0,label1,windowtext
windowtext  dc.b  "Mein erstes Window",0
```

In diesem Fall würde ein Window eröffnet, das 320*256 Punkt groß ist. Durch Setzen der IDCMP-Bits 8 und 9 bekämen wir Nachricht bei Anwahl eines Menüpunktes und des Close-Gadgets. Da wir keine eigenen Gadgets benutzen wollen, folgt eine Null, danach ein beliebiger Labelname, der die Adresse des Langwortes definiert, in das wir den Screenpointer eintragen müssen. Als letztes folgt dann der Zeiger auf den Window-Titel.

2.2.2 Die Window-Struktur

Nachdem wir nun die NewWindow-Struktur besprochen haben, lassen Sie uns zu der Window-Struktur überwechseln. Diese wird beim Öffnen des Windows von Intuition angelegt. Den Zeiger erhalten wir nach dem Aufruf zurück.

Offset	Typ	Name	Funktion
$000	L	Next Window	Zeiger auf nächste Fenster-Struktur
$004	W	LeftEdge	Linke Ecke (x-Koordinate)
$006	W	TopEdge	Obere Ecke (y-Koordinate)
$008	W	Widht	Breite in Pixel
$00A	W	Height	Höhe in Pixel
$00C	W	MouseY	Mausposition (y-Koordinate)
$00E	W	MouseX	Mausposition (x-Koordinate)
$010	W	MinWidht	Minimale Breite in Pixel
$012	W	MinHeight	Minimale Höhe in Pixel
$014	W	MaxWidht	Maximale Breite in Pixel
$016	W	MaxHeight	Maximale Höhe in Pixel
$018	L	Flags	Window-Flags (wie in NewWindow)
$01C	L	MenuStrip	Zeiger auf Menüstrukur
$020	L	Title	Zeiger auf Window-Titel
$024	L	FirstRequest	Zeiger auf ersten Requester
$028	L	DMRequest	Zeiger auf DM-Requester
$02C	W	ReqCount	Anzahl Requester
$02E	L	WScreen	Zeiger auf Screen-Struktur
$032	L	RPort	Zeiger auf Rast-Port (s.u.)
$036	B	BorderTop	Dimension der Titelleiste
$037	B	BorderLeft	des Windows
$038	B	BorderRight	
$039	B	BorderBottom	
$03A	L	BorderRPort	Zeiger auf Rast-Port Gimmezerozero
$03E	L	FirstGadget	Zeiger auf erste Gadget-Struktur
$042	L	Parent	Zeiger auf vorheriges und
$046	L	Descendant	folgendes Window
$04A	L	Pointer	Zeiger auf Spritedaten vom Mauszeiger
$04E	B	PtrHeight	Mauszeiger Höhe
$04F	B	PtrWidht	Mauszeiger Breite
$050	B	XOffset	Koordinaten des wirklichen
$051	B	YOffset	Zeigerpunktes (z.B. Pfeilspitze)
$052	L	IDCMPFlags	wie in NewWindow-Struktur

Offset	Typ	Name	Funktion
$056	L	UserPort	Zeiger auf User-Port (s.u.)
$05A	L	WindowPort	Zeiger auf Window-Port (s.u.)
$05E	L	MessageKey	Zeiger auf Intuition-Message
$062	B	DetailPen	Farben wie in NewWindow-
$063	B	BlockPen	Struktur
$064	L	CheckMark	Zeiger auf Grafikdaten Mauszeiger
$068	L	ScreenTitle	Zeiger auf Screentitel
$06C	W	GZZMouseX	Mausposition im
$06E	W	GZZMouseY	Gimmezerozero-Window
$070	W	GZZWidht	Breite und Höhe des
$072	W	GZZHeight	Gimmezerozero-Windows
$074	L	ExtData	Zwei User-Zeiger zur
$078	L	UserData	beliebigen Anwendung
$07C	L	WLayer	Zeiger auf Layer-Struktur des Windows

Nun zu den Funktionen der drei Ports: der wichtigste ist der sogenannte Rast-Port. Er stellt die Schnittstelle zu den Grafik-Kernroutinen des Systems dar. Da beim Amiga nur ein Grafikmodus existiert (kein Textmodus wie z.B. beim C64), wird alles, was auf dem Bildschirm erscheint, durch diese Grafikroutinen aufgebaut. Der Rast-Port ist daher unverzichtbarer Bestandteil der Programmierung.

Ebenso bedeutsam sind der Window- und User-Port. Der User-Port wird dazu verwendet, Nachrichten zu empfangen, der Window-Port dient zum Senden einer Nachricht. Wenn Sie ein Window nicht mehr benötigen, sollten Sie es sofort schließen, um dem System den wertvollen Speicherplatz zurückzugeben. Theoretisch könnten Sie natürlich auch beliebig viele Windows überlappen, irgendwann jedoch wird Ihr RAM einen Kollaps erleiden und den Guru um Hilfe bitten müssen. Daher benutzen Sie besser die folgende Routine:

```
move.l windowpointer,a0
CALLINT CloseWindow
```

2.2.3 Modifikation des Mauszeigers

Auch bezüglich der Windows existieren viele Funktionen, die Sie ausführlich im Anhang aufgeführt sehen. Die beiden interessantesten möchte ich Ihnen an dieser Stelle vorstellen:

```
SetPointer
```

Diese Funktion dient dazu, den Mauszeiger zu modifizieren. Er kann bis zu 16 Pixel breit und beliebig hoch sein. Dazu ist es notwendig, die neuen Grafikdaten in einem sogenannten Image zu definieren, dessen Aufbau wir noch besprechen werden. Zunächst ist es nur wichtig zu wissen, wie die Funktion aufgerufen wird:

```
move.l windowpointer,a0
move.l #image,a1             ;Zeiger auf neue Grafikdaten
move.l #höhe,d0              ;Höhe in Pixel
move.l #breite,d1            ;Breite in Pixel
move.l #xoffset,d2           ;"Hot Point" (x-Koordinate)
move.l #yoffset,d3           ;"Hot Point" (y-Koordinate)
CALLINT SetPointer
```

Der sogenannte »Hot Point« stellt denjenigen Punkt innerhalb des Mauszeigers dar, der für das Anklicken der Funktionen relevant ist, so z.B. bei dem Standardpfeil die Pfeilspitze. Bitte beachten Sie, daß die Offsets relativ zu der rechten unteren Ecke der Grafik gezählt werden, d.h. immer negativ sein müssen, damit der »Hot Point« innerhalb der Grafik liegt! Ein positiver Wert bedeutet immer, daß der Punkt außerhalb der Grafik liegt. Der neue Mauszeiger ist immer dann zugegen, wenn das entsprechende Window (d.h. das Window, dessen Pointer in Windowpointer abgelegt ist) aktiviert ist.

```
ClearPointer
```

Diese Funktion aktiviert wieder den Standardpfeil. Sie wird aufgerufen mit:

```
move.l windowpointer,a0
CALLINT ClearPointer
```

2.3 Die Menüs

Die Pull-down-Menüs gehören zu den angenehmsten Eigenschaften von Intuition. Sie bieten eine ungeheure Vielfalt an Variationen an, die man auf den ersten Blick kaum erkennen kann. Da die Kontrolle wie immer natürlich bei Intuition liegt, kann man ahnen, daß wiederum eine Menge an Parametern in einer Struktur übergeben werden müssen. Bevor wir uns unseren Menü-Makros zuwenden, wollen wir jedoch zunächst einmal den prinzipiellen Aufbau eines Menüs kennenlernen. Dazu nehmen wir ein Bildschirmfoto eines Beispielprogramms dieses Buches, der Wertetabelle, zur Hand (Bild 2.5).

Wenn man die rechte Maustaste betätigt, erscheint am oberen Rand des Screens ein Streifen mit drei Menütiteln. Diesen Streifen nennt man Menu-Strip. Wenn man die Maustaste wieder losläßt, verschwindet der Streifen wieder. Es ist wichtig zu wissen, daß ein Menü immer zu einem Window gehört und nur dann aktiviert werden kann, wenn auch das Window aktiv ist. Wenn man dann den Mauszeiger auf einen dieser Menütitel positioniert, wird das Pull-down-Menü heruntergeklappt und die einzelnen Menüpunkte (Items) erscheinen. Durch weitere Bewegungen des Mauszeigers auf die Items werden diese nun jeweils hervorgehoben (z.B. invertiert wie in unserem Fall), eventuell erscheint ein Untermenü mit weiteren Untermenüpunkten (Sub-Items). Dieses ist nun die letzte Ebene, die von Intuition verwaltet werden kann. Es ist nicht sinnvoll, ein Unter-Untermenü zu erzeugen, da man keinen Zugriff auf die einzelnen Unter-Untermenüpunkte mehr hat, obwohl eine weitere Verzweigung technisch durchaus realisierbar wäre. Bis jetzt ist es für

Sie nur wichtig zu wissen, was man sich unter den Begriffen Menue-Strip, Item und Sub-Item vorzustellen hat. Wie man diese Struktur technisch aufbaut, werden wir nun besprechen.

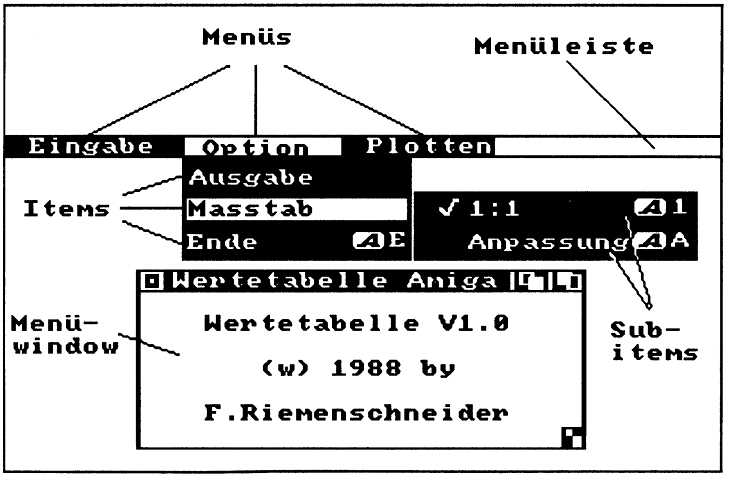


Bild 2.5: Aufbau einer Menüstruktur

2.3.1 Die Struktur des Menue-Strip

Um einem Fenster eine Menüleiste zuzuordnen, ist wie immer nur ein einfacher Befehl notwendig:

```
move.l windowpointer,a0
move.l #menu,a1
CALLINT SetMenuStrip
```

Dabei ist »windowpointer« der Pointer auf die Window-Struktur des Fensters, dem der Menue-Strip zugeordnet werden soll. Bei Menue handelt es sich um einen Zeiger auf eine Struktur, in der die wesentlichen Informationen über das erste Menü zu finden sind. Ein Makro für diese Struktur sieht so aus:

```
*Struktur für Menü

MENUE           macro
dc.l \1         ;nächstes Menü
dc.w \2,\3      ;x,y des Menütitels
dc.w 70,10      ;Breite,Höhe des Menütitels
dc.w 1          ;Menü wählbar
dc.l \4         ;Menütitel
dc.l \5         ;Menüeinträge
dc.w 0,0,0,0
endm
```

Ein Menue-Strip besteht im allgemeinen ja nicht nur aus einem Menü, sondern aus mehreren, wie drei in unserem Beispiel. Daher enthält die Menü-Struktur als erstes einen Zeiger auf die nächste, natürlich vom Aufbau identische, Menü-Struktur. Es folgen die Koordinaten, die die Startposition innerhalb der Menüleiste festlegen. Es folgen Breite und Höhe des Menütitels. Man sollte diese immer so dimensionieren, daß bei der Anwahl auch wirklich nur der Menütitel invertiert wird und nicht die halbe Menüleiste.

Nun kommt ein Flag, das anzeigt, ob das Menü wählbar ist. In diesem Fall ist eine Eins einzutragen. Möchte man das Menü unwählbar machen, erscheint dies gepunktet, wie Sie es sicherlich auch schon von der Workbench kennen. Sie können mit dem Mauszeiger soviel herumfahren, wie Sie wollen, es passiert nichts. Über den Sinn oder Unsinn einer solchen Maßnahme werden wir noch sprechen. Es folgt ein Zeiger auf einen String, der durch eine Null abgeschlossen werden muß und den Menütitel repräsentiert.

Als letztes folgt ein Zeiger auf eine weitere Struktur, die zu dem ersten Item gehört. Diese ist von der Menü-Struktur natürlich grundsätzlich verschieden aufgebaut, so daß man beide auf keinen Fall verwechseln darf. Der Guru würde für lange Zeit ins Grübeln kommen. Die weiteren vier Worte werden von Intuition beschrieben.

2.3.2 Aufbau der Item- und Subitem-Struktur

Nun kommen wir zu der Struktur der Items und Subitems. Sie ist in beiden Fällen identisch aufgebaut, so daß wir nur ein Makro benötigen:

```
*Struktur für Menü- und Untermenüpunkte
MENUPUNKT         macro
dc.l  \1          ;nächster Menüpunkt
dc.w  \2,\3       ;x,y-Position
dc.w  \4          ;Breite
dc.w  10          ;Höhe
dc.w  \5          ;Flag
dc.l  \6          ;Ausschluß
dc.l  \7          ;Zeiger auf Textstruktur
dc.l  0           ;keine Zeichnung
dc.b  \8          ;Amiga-Taste
even
dc.l  \9          ;Zeiger auf Untermenü
dc.l  0
endm
```

Da die Einträge der Items und Subitems jedoch in diversen Punkten differieren, wollen wir uns zunächst auf die Items beschränken. Als erstes haben wir wieder einen Zeiger auf die Struktur des folgenden Menüpunktes. Wenn wir keinen mehr möchten, tragen wir, wie auch in der Menüstruktur, eine Null ein. Nun folgen x- und y-Position relativ zur Lage des Menütitels, gefolgt von Breite und Höhe. Intuition dimensioniert den rechteckigen Kasten um die Menüpunkte immer so, daß der breiteste Menüpunkt immer noch hineinpaßt.

Danach kommt der interessanteste Punkt der Struktur: Ein Flag, durch das wir das Aussehen und die Funktion unseres Menüpunktes sehr variabel gestalten können.

Bit	**Wert**	**Name**
0	$0001	CHECKIT

Dieses Bit unterscheidet zwischen den sogenannten Verb-Items und den Attribut-Items. Während ein Verb-Item eine bestimmte Aktion veranlassen soll, wie z.B. eine Ausgabe oder das Programmende im Bild 2.5, werden durch Attribut-Items bestimmte Zustände charakterisiert. In unserem Beispiel kann man den Maßstab der Grafikausgabe festlegen. Durch ein Attribut-Item wird also nie eine bestimmte Aktion ausgelöst, sondern irgendein Programm-Parameter festgelegt. Deshalb bietet Intuition die Möglichkeit, ein solches Item durch einen Haken kenntlich zu machen. Wenn Sie also das 0. Bit setzen, wird der Menüpunkt abgehakt oder mit einer anderen Grafik versehen, die Sie beim Öffnen des Fensters festgelegt haben. Man muß beachten, daß die Startposition des Items nach rechts verschoben werden muß, damit davor noch Platz für den Haken ist. Sie sollten die x-Koordinate hierfür um mindestens 10–12 Punkte verschieben, um ein Überschreiben des Items zu verhindern.

Bit	**Wert**	**Name**
1	$0002	ITEMTEXT

Wenn Sie dieses Bit setzen, wird das Item in Textform dargestellt. Im anderen Fall wird eine selbstdefinierte Grafik ausgegeben. Ein Programm, in dem von dieser Möglichkeit Gebrauch gemacht wird, ist DeluxePaint 2. Die Grafik wird dabei wie der Mauszeiger durch ein Image dargestellt.

Bit	**Wert**	**Name**
2	$0004	COMMSEQ

Durch Setzen dieses Bits wird die Anwahl des Items nicht nur durch die Maus, sondern auch durch die Kombination der rechten Amiga-Taste mit einer weiteren beliebigen Taste ermöglicht, deren ASCII-Code man ebenfalls in die Item-Struktur eintragen muß (siehe unten). In dem Menü erscheint hinter dem Item die wunderschöne Grafik der Amiga-Taste, so daß man wiederum Platz reservieren muß, um ein Überschreiben des Items zu verhindern. Diesmal muß jedoch rechts vom Text reserviert werden, so daß die Breite des Items um ca. 20 Punkte vergrößert werden sollte. Ein Punkt ist jedoch ganz wichtig: Bitte lassen Sie es niemals zu, daß kritische Operationen wie z.B. das Formatieren einer Diskette durch eine Tastenkombination ermöglicht werden. Man kann sich sehr schnell einmal vertippen und wertvolle Daten unbeabsichtigt für immer vernichten.

Bit	**Wert**	**Name**
3	$0008	MENUTOGGLE

Hierbei wird der Haken (oder die selbstdefinierte Grafik) wechselweise ein- und ausgeschaltet. Sinnvoll ist diese Operation z.B. bei der Festsetzung des Schriftmodus: Hat man einen Menüpunkt »Text unterstreichen« , so wird dieser Modus bei der ersten Anwahl eingeschaltet und somit abgehakt. Bei einer weiteren Anwahl wird der Modus wieder

gelöscht und der Haken verschwindet wieder. In unserem zweiten Beispielprogramm werden wir von dieser Möglichkeit Gebrauch machen.

Bit	**Wert**	**Name**
4	$0010	ITEMENABLED

Durch dieses Bit wird festgelegt, ob der Menüpunkt wählbar ist. Ist es nicht gesetzt, erscheint der Menüpunkt punktiert und reagiert auf keine Maus und schon gar nicht auf eine Amiga-Tastenkombination. Bedeutsam kann dieser Modus dann werden, wenn ein Item nur in einer bestimmten Situation sinnvoll erscheint, z.B. ist bei einem Textprogramm eine Blockoperation nur dann möglich, wenn vorher ein Block markiert wurde. Man könnte das Item der Blockoperation daher so lange unwählbar machen, bis ein Block markiert wurde. Alle Subitems werden automatisch ebenfalls unwählbar gemacht.

Bit	**Wert**	**Name**
6/7	$0040	HIGHCOMP
	$0080	HIGHBOX
	$0000	HIGHIMAGE
	$00C0	HIGHNONE

Diese beiden Bits legen fest, auf welche Weise ein Item bei Berührung mit der Maus hervorgehoben wird. Während HIGHCOMP das Item invertiert, wird es durch HIGHBOX mit einem Rahmen versehen. HIGHNONE unterdrückt jede Reaktion und HIGHIMAGE läßt ein weiteres Image erscheinen.

Bit	**Wert**	**Name**
8	$0100	CHECKED

Dieses Bit wird sowohl von Intuition als auch vom Programm gesetzt. Intuition setzt es immer dann, wenn Sie einen Menüpunkt angewählt haben, der mit einem Haken versehen werden soll, bei dem also auch CHECKIT gesetzt ist. Man kann auf diese Weise testen, ob ein bestimmtes Item abgehakt ist. Wenn man das Bit vom Programm aus setzt, wird das entsprechende Item zwangsläufig abgehakt, egal ob man es mit der Maus oder einer Tastenkombination angewählt hat oder nicht.

In unserem Makro geht es nun mit einem Langwort weiter, das den gegenseitigen Ausschluß von Items regelt. Nehmen wir einmal an, Sie haben ein Menü, in dem jedes Item eine bestimmte Zeichenfarbe repräsentiert. Es ist unmittelbar einsichtig, daß Sie zu jeder Zeit nur eine Farbe gleichzeitig benutzen können. Das Problem für Sie als Programmierer besteht nun darin, daß Sie nicht nur bei der Anwahl einer Farbe diese abhaken müssen (dieses tut Intuition für Sie), sondern daß Sie theoretisch die Haken aller anderen Farben entfernen müßten. Dies würde über das Flag CHECKED möglich sein. Dieser Weg wäre jedoch sehr umständlich. Daher bietet Intuition die Möglichkeit, bei Anwahl eines bestimmten Punktes die Bits CHECKED von weiteren Items, die Sie festlegen können, automatisch zu löschen und damit auch die Haken zum Verschwinden zu bringen. Die Items sind dafür bitweise numeriert worden.

Bit	Wert	Item
0	1	1
1	2	2
2	4	3
3	8	4
usw.		

Wenn Sie nun z.B. in die Struktur des Items 1 den Wert 10 eintragen würden, wären damit die Items 4 und 2 ausgeschlossen worden, d.h., wann immer Sie das Item 1 anwählen, würden die Haken der Items 4 und 2 sofort verschwinden. Falls Sie alle weiteren Items bis auf das angewählte Item selbst ausschließen möchten, tragen Sie einfach nur den Wert $FFFF minus den Wert des Items selbst ein, z.B. $FFFE bei Item 1, $FFFD bei Item 2 usw. Es spielt dabei keine Rolle, daß auch Items ausgeschlossen werden, die gar nicht existieren, ausnahmsweise ist der Guru gnädig mit uns.

Nun folgt ein weiterer Zeiger auf die Textstruktur des Items, wenn das Flag ITEMTEXT gesetzt wurde. Im anderen Fall muß der Zeiger auf die selbstdefinierte Zeichnung gerichtet sein. Das nächste Langwort ist nur dann von Bedeutung, wenn das HIGHIMAGE-Flag gesetzt wurde. Es enthält dann den Zeiger auf die Grafikdaten. In der Regel kann man daher eine Null eintragen. Danach kommt ein Byte, das den ASCII-Code der Taste enthält, die zusammen mit der rechten Amiga-Taste die Anwahl eines Items zuläßt. Natürlich ist ein Eintrag nur dann sinnvoll, wenn auch das COMMSEQ-Flag gesetzt wurde.

Da als nächstes wieder ein Langwort folgt, muß man zunächst mit EVEN die Adresse begradigen (wehe, nicht...). Das Langwort enthält einen Zeiger auf die Struktur des ersten Subitems, das zu dem angewählten Item gehört. Bei der Anwahl eines Items, welches ein oder mehrere Subitems besitzt, werden diese automatisch in einer weiteren Box angezeigt. Durch Verschieben des Mauszeigers in diese Box kann man dann die entsprechenden Subitems anwählen. Möchte man keine Subitems, trägt man eine Null ein.

Das letzte Langwort enthält eine interessante Funktion, die wir in unseren Beispielprogrammen jedoch nicht benutzen werden: Die Anwahl mehrerer Items gleichzeitig. In dem Langwort wird von Intuition die codierte Menünummer (siehe unten) des zuletzt angewählten Items gespeichert. In der Praxis hat man zwei Möglichkeiten, mehrere Punkte anzuwählen: Zum einen kann man bei gedrückter rechter Maustaste mehrere Items hintereinander mit der linken Maustaste anklicken, zum anderen kann man aber auch einfach beide Tasten gedrückt halten und dann mit der Maus über mehrere Items hinwegfahren. Um diese Items abzufragen, müßte man solange die Menü-Events auswerten (Kapitel 2.5), bis das Langwort wieder den Wert Null annimmt.

Wie oben schon erwähnt wurde, ist die Struktur des Subitems mit der eines Items identisch. Man kann prinzipiell auch alle Funktionen übernehmen, bis auf den Zeiger auf ein weiteres Untermenü. Dieser ist nämlich nicht zulässig, da ein Unter-Untermenü nicht mehr

ausgewertet werden kann. Man muß hier also immer eine Null eintragen. Der erste Zeiger, der bei den Items auf das nächste Item zeigt, dient bei den Subitems analog dazu, auf das folgende Subitem zu zeigen. Möchte man kein weiteres Subitem, ist auch hier eine Null einzutragen. Um Ihnen zum Schluß noch einmal einen Überblick zu geben, habe ich die Menüstruktur aus Bild 2.5 in Bild 2.6 einmal so aufgeschlüsselt, wie sie von Intuition gesehen wird.

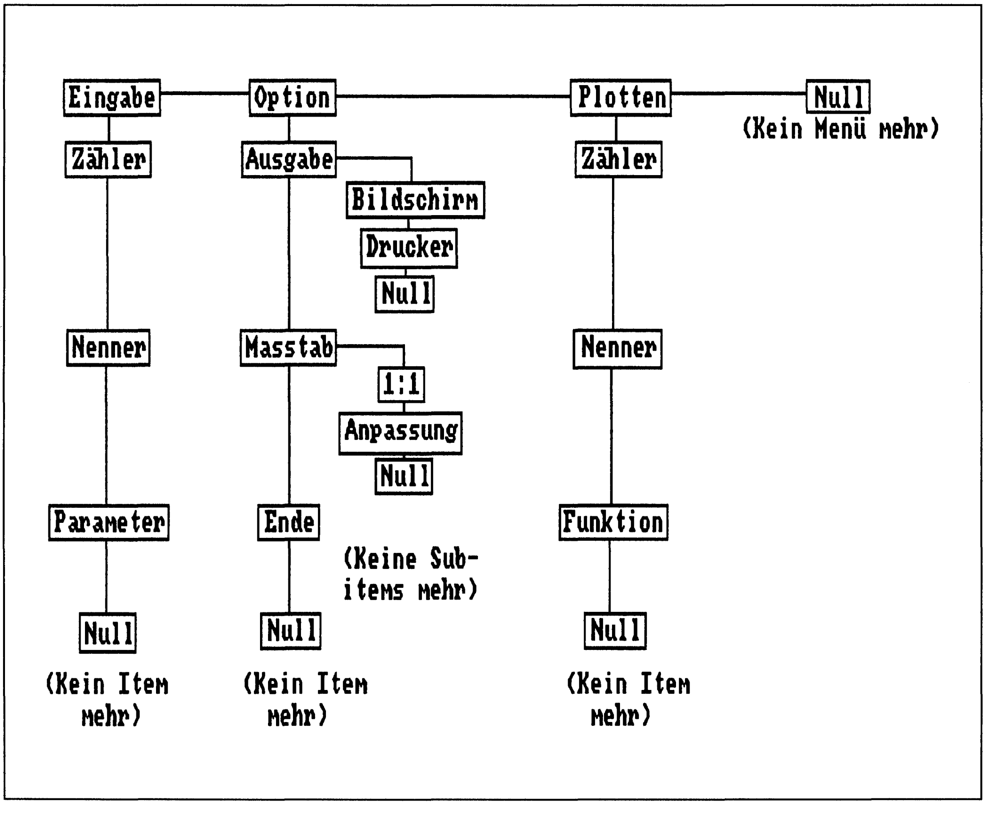


Bild 2.6: Menüstruktur aus der Sicht von INTUITION

2.3.3 Menüs wählbar und unwählbar machen

Durch das Flag-Bit ITEMENABLED wird nur der Anfangszustand des Items bestimmt. Oft ist es jedoch erforderlich, ein Item während des Programmlaufes wählbar oder unwählbar zu machen. Dazu dienen die Funktionen Offmenu (Unwählbar machen) und Onmenu (wählbar machen). Aufgerufen werden sie wie folgt:

```
move.l windowpointer,a0
move.l code,d0
CALLINT Off(On)Menu
```

Sie werden sich wahrscheinlich wundern, daß neben dem Windowpointer nur ein Langwort übergeben wird. Dabei sind doch drei Parameter erforderlich: die Menünummer, die Itemnummer und die Subitemnummer. Sie haben es wahrscheinlich schon an Hand des Langwortnamens Code erahnt, daß alle drei Werte in dem Langwort enthalten sind. Dabei gilt:

Bit 0.. 4 : Menünummer
Bit 5..10 : Itemnummer
Bit 11..15 : Subitemnummer

Damit kann man maximal 31 Menüs mit jeweils 63 Items codieren, die ihrerseits wiederum jeweils 31 Subitems enthalten dürfen. Da die Abfrage des gewählten Items ebenfalls in diesem Codelangwort erfolgt (Kapitel 2.5.), sind dies die maximal zulässigen Werte, da größere Zahlen nicht verwaltet werden können. Es handelt sich jedoch um mehr theoretische Werte, da die Anzahl der Menüs und ihrer Items schon durch den Bildschirm begrenzt werden.

Ich möchte Ihnen nun eine universelle Routine vorstellen, die auf einen Schlag sämtliche Menüs samt ihrer Inhalte (un)wählbar macht. Dies ist in der Praxis oft erforderlich, z.B. immer dann, wenn Ihr Programm mit längeren Operationen wie Druckerausgaben etc. beschäftigt ist. Der DEVPAC-Assembler macht so während des Assemblierens alle Menüs unwählbar.

```
        move.l #0,d4                    ;Menü
einl    move.l #0,d3                    ;Menüpunkt
einl1   move.l #0,d5                    ;Untermenüpunkt
        move.l d3,d2                    ;Menüpunkt nach Bit 5-10
        lsl.l #5,d2
einl2   move.l d5,d0                    ;Untermenüpunkt nach Bit
        lsl.l #7,d0                     ;11-15
        lsl.l #4,d0
        add.l d2,d0                     ;Menüpunkt einblenden
        add.l d4,d0                     ;Menü einblenden
        move.l windowpointer,a0         ;Menüpunkt
        CALLINT (Off)OnMenu             ;(un)wählbar machen
        add.l #1,d5
        cmp.l #subitems+1,d5            ;Anzahl Untermenüs+1
```

```
bne einl2
add.l #1,d3
cmp.l #items+1,d3              ;Anzahl Items+1
bne einl1
add.l #1,d4
cmp.l #menus,d4                ;Anzahl Menüs
bne einl
```

In die Variable »Menus« müssen Sie die Anzahl der verwendeten Menüs eintragen, in die Variable Items die maximal in einem Menü vorkommende Zahl an Items sowie die maximal auftretende Anzahl von Subitems in die gleichnamige Variable.

Wie arbeitet diese Routine? Nun, sie besitzt drei ineinander verschachtelte Schleifen. Die innerste Schleife repräsentiert die Untermenüs. Dabei wird der Schleifenwert immer um 11 Bit nach links verschoben. Wenn alle Subitems eines bestimmten Items bearbeitet wurden, wird der Itemwert um eins erhöht, bis auf diese Weise auch alle Items behandelt wurden. Dabei wird der Itemwert um 5 Bit nach links verschoben. In der äußersten Schleife schließlich wird die Menünummer jeweils um eins erhöht. Die drei Werte werden jeweils addiert, um das Codewort zu bilden, das dann der Routine übergeben wird.

Etwas merkwürdig scheint die Tatsache zu sein, daß bei den Items und Subitems ein Wert mehr bearbeitet werden muß, als tatsächlich vorhanden ist. Ich kann Ihnen nicht verraten, warum dies so ist, aber was passiert, wenn man sich nicht nach dieser »Vorschrift« richtet: Falls Sie das nicht existierende, letzte Subitem nicht ansprechen, werden Sie erleben, daß das zugehörige Item nicht reagiert, d.h., alle Subitems können unwählbar sein und auch punktiert dargestellt werden, das Item jedoch erscheint nach wie vor als wählbar. Erst wenn Sie das fiktive Subitem unwählbar gemacht haben, wird auch das Item unwählbar. Genauso verhält es sich mit den Items und den Menüs: Erst das Unwählbarmachen des fiktiven Items macht das ganze Menü unwählbar. Umgedreht gilt die ganze Sache natürlich auch. Vorsicht ist aber bei den Menüs angesagt: Hier muß unbedingt die richtige Anzahl eingesetzt werden, sonst wird wieder meditiert! Sind das nicht schreckliche Zustände, liebe Leser? Ich jedenfalls halte von solchen »Gags« des Betriebssystems herzlich wenig.

Interessant ist jedoch, daß man soviel nicht existierende Items und Subitems ansprechen kann, wie man Lust hat. In diesem Fall tritt weder der Guru noch irgendeine andere unangenehme Überraschung in Aktion.

2.4 Die Gadgets

Den Begriff Gadget (Dingsda) haben wir in den letzten Abschnitten schon öfter verwendet. Allerdings handelte es sich hierbei um die sogenannten Systemgadgets, wie z.B. das Fensterschließsymbol. Daneben ist es aber auch möglich, eigene, sogenannte Anwendergadgets zu definieren, die zu den verschiedensten Zwecken eingesetzt werden können.

Prinzipiell ist ein Gadget nichts anderes als ein besonderes, normalerweise unsichtbares Rechteck auf dem Bildschirm, das auf Anklicken mit der Maus eine Reaktion auslöst. Um

ein Gadget sichtbar zu machen, kann man es umrahmen, einen Text ausgeben oder ein Image darstellen. Diese Möglichkeiten sind auch kombinierbar. Ein Anwender-Gadget ist immer einem Window zugeordnet, obwohl die Gadgets an sich auch z.B. in Screens oder Requestern (Kapitel 2.10) erscheinen können. Man kann zwischen vier Gadget-Typen unterscheiden, wenn auch der vierte Typ ein Sonderfall des dritten ist (siehe unten):

1) **Die Boolean-Gadgets**
Hierbei handelt es sich um den primitivsten Typ. Er kann nur, wie der Name auch sagt, zwischen zwei Zuständen unterscheiden: Entweder ist das Gadget durch das Anklicken mit der Maus aktiviert worden, oder es ist eben nicht aktiv. Ein solches Gadget eignet sich für Abfragen beliebiger Art wie z.B. »OK« und »Cancel«.

2) **Die Proportional-Gadgets**
Dieser Typ stellt Schieberegler dar, wie sie z.B. in Preferences für die Auswahl der Farbanteile verwendet werden. Der Schieberegler kann mit der Maus bewegt werden, je nach Stellung erhält der Anwender einen dieser Stellung entsprechenden Wert zurück.

3) **Die String-Gadgets**
Hierbei handelt es sich um Eingabefelder für Texte, die man auch noch komfortabel editieren kann.

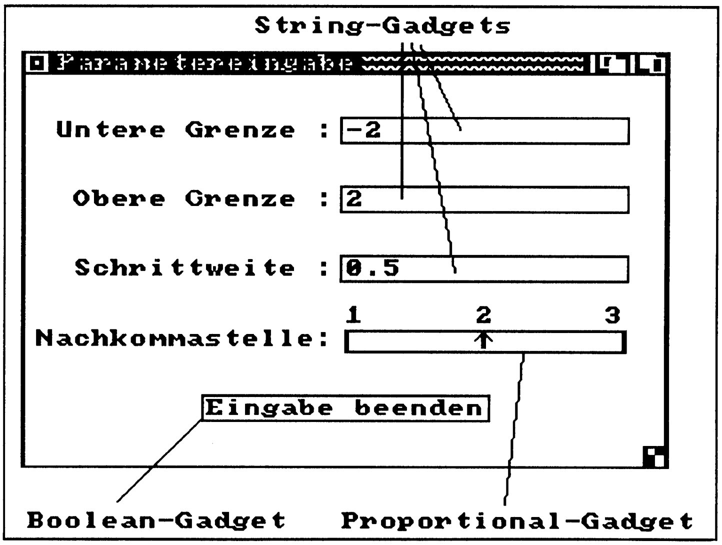


Bild 2.7: Diverse Gadget-Typen

4) **Die Integer-Gadgets**
Sie stellen einen Sonderfall der String-Gadgets dar. Ein Integer-Gadget erlaubt nämlich nur numerische Eingaben und das Vorzeichen, alle anderen Eingabeversuche werden unterbunden. Das Ergebnis wird automatisch in ein Langwort umgerechnet und dem Anwender zur Verfügung gestellt. Leider ist es jedoch nicht möglich, beliebige Fließkommazahlen einzugeben.

Man sollte sich angewöhnen, sein eigenes Programm mit möglichst vielen Gadgets auszustatten, da sie einerseits eine komfortable Eingabemöglichkeit bieten und andererseits viel Arbeit sparen, auch wenn das Einrichten eines Gadgets wie immer mit viel Fleißarbeit verbunden ist.

Die Gadget-Struktur

Da, wie gesagt, die Grundstruktur aller Gadgets identisch ist, kann man sie wiederum mit einem Makro erschlagen:

```
* Struktur für Gadgets
GADGET   macro
dc.l       \1   ;nächstes Gadget
dc.w       \2   ;obere Ecke x
dc.w       \3   ;obere Ecke y
dc.w      122   ;Breite
dc.w       10   ;Höhe
dc.w       \4   ;Flags
dc.w        2   ;Activation-Flags
dc.w       \5   ;Gadget-Typ:String
dc.l       \6   ;Gadget-Border
dc.l        0   ;kein Select-Gadget
dc.l       \7   ;Gadget-Text
dc.l        0   ;kein Exclude
dc.l       \8   ;Special-Info
dc.w       \9   ;Gadget-ID
dc.l        0   ;User-Data
endm
```

Das erste Langwort dient dazu, einen Zeiger auf ein etwaiges weiteres Gadget aufzunehmen. Die Anzahl der Gadgets in einem Window ist theoretisch unbegrenzt, aber irgendwann wird zunächst der Platz auf dem Bildschirm und dann auch des Speichers zur Neige gehen.

Nun folgen die Positionsangaben relativ zur linken oberen Ecke des Windows. Es ist durch Setzen einiger Flags (siehe unten) jedoch möglich, die Lage eines Gadgets der des Windows anzupassen, dazu jedoch später.

Als nächstes folgen zwei Wörter, welche die Größe des Gadgets bestimmen. Dabei handelt es sich jedoch nicht, wie oft fälschlich angenommen wird, um die sichtbaren Ausmaße des Gadgets, sondern vielmehr um die Größe des (zunächst unsichtbaren) Rechtecks, welches von Intuition überwacht wird. Sicherlich ist es aber später sinnvoll, genau diesen Ausschnitt auch sichtbar zu machen. Das Anklicken eines Bildschirmpunktes außerhalb dieses Rechtecks führt zu keiner Reaktion.

Nun kommt ein Wort, dessen Bits bestimmen, was für eine Art von Reaktion auf das Anklicken des Gadgets folgen soll und ob relative Koordinaten eingeführt werden. Zunächst zu der Reaktion. Hierfür sind die Bits 0 und 1 zuständig:

Bit 1	0	Wert	Name	Bedeutung
0	0	0	GADGHCOMP	Das Gadget wird komplementiert
0	1	1	GADGHBOX	Das Gadget wird umrandet
1	0	2	GADGHIMAGE	Es erscheint ein anderes Image
1	1	3	GADGHNONE	Keine Reaktion

Sie können bei der Anwahl des Gadgets durch Setzen des Bit 0 also dessen Aussehen verändern. Wenn dann ein anderes Image erscheinen soll, müssen Sie zusätzlich Bit 2 (GADGIMAGE) setzen. Durch ein nicht gesetztes Bit 2 wird Intuition mitgeteilt, daß ein Border (rechteckige Box, Kapitel 2.9.) erscheinen soll.

Die nun folgenden 4 Bit ermöglichen es, daß ein Gadget an einer Kante »klebt« und Höhen- sowie Breitenangaben relativ zu dieser Kante verstanden werden. Das Setzen von Bit 3 (GRELBOTTOM) führt dazu, daß die Position des Gadgets nicht mehr relativ zur Oberkante des Windows, sondern zu dessen Unterkante gesehen wird. Die Folge davon ist, daß bei einer Vergrößerung des Windows in y-Richtung das Gadget mitwandert, wie es z.B. das Größen-Gadget der Windows tut. Natürlich muß die Lage des Gadgets durch eine negative Koordinate beschrieben werden, da es ja oberhalb der Bezugskante startet.

Durch Bit 4 (GRELRIGHT) wird derselbe Effekt auf horizontaler Ebene erzielt. Offensichtlich ist bei dem Größen-Gadget auch dieses Bit gesetzt, denn bei einer Vergrößerung des Windows in x-Richtung wandert es ja ebenfalls mit. Selbstverständlich muß auch die x-Position in diesem Fall durch einen negativen Wert beschrieben werden, da die rechte Kante des Windows die Bezugskante ist.

Bit 5 (GRELWIDHT) führt dazu, daß zur in der Gadget-Struktur definierten Breite immer die aktuelle Windowbreite hinzuaddiert wird. Dieser Effekt wird z.B. beim Drag-Gadget ausgenutzt. In der Struktur wurde für die Breite eine Null eingetragen, so daß das Gadget immer die Breite des Windows ausmacht.

Schließlich wird durch Setzen des Bit 6 (GRELHIGHT) zur Standard-Höhe des Gadgets immer die aktuelle Window-Höhe hinzuaddiert. Falls man eines der letzten beiden Bits setzt, muß man sich im klaren darüber sein, daß als größter zulässiger Wert für Höhe und Breite eine Null in die Gadget-Struktur eingetragen werden darf, da das Gadget die Ausmaße des Windows nicht überschreiten darf. Sinnvoll sind im allgemeinen nur negative Werte, die dazu führen, daß das Gadget kleiner als das Window dargestellt wird.

Die letzten beiden Bits geben Auskunft über die Gadget-Aktivität: Falls Bit 7 (SELECTED) gesetzt wird, wird das Gadget sofort nach Öffnen des Fensters aktiv, braucht also nicht zuerst mit der Maus angeklickt zu werden. Dies ist immer dann sinnvoll, wenn

z.B. nur ein Gadget vorhanden ist. Schließlich macht Bit 8 (GADGDISABLED) das entsprechende Gadget unwählbar. Falls Sie mehrere Flags gleichzeitig setzen möchten, müssen Sie wie immer die entsprechenden Bit-Werte addieren. Nun folgt wiederum ein Wort mit Flags, die durch die einzelnen Bits repräsentiert werden. Sie nennen sich Activision-Flags und bestimmen diverse Funktionen der Gadgets:

Bit	Wert	Name	Bedeutung
0	1	RELVERIFY	Gadget wird nur dann aktiviert, wenn der linke Mausknopf auch über dem Gadget losgelassen wird
1	2	GADGIMMEDIATE	Bei Aktivierung des Gadgets wird eine Nachricht gesendet.
2	4	ENDGADGET	Läßt Gadget in Requester diesen bei Anwahl beenden und damit verschwinden
3	8	FOLLOWMOUSE	Wirkt wie REPORTMOUSE bei Windows, indem bei angewähltem Gadget so lange die Mausposition gemeldet wird, bis das Gadget deaktiviert wird.
4	16	RIGHTBORDER	
5	32	LEFTBORDER	
6	64	TOPBORDER	
7	128	BOTTOMBORDER	

Zu den letzten vier Flags ist eine etwas ausführlichere Erklärung notwendig. Wenn Sie schon einmal einen Blick auf das übernächste Langwort in unserem Makro richten, sehen Sie, daß dieses einen Zeiger auf eine Border- oder Image-Struktur enthält. Das Rechteck oder die Grafik erscheinen beim Aufbau des Gadgets und dienen dazu, es sichtbar zu machen. Bei der Verwendung von relativer Höhe und Breite tritt jedoch das Problem auf, daß sich die Größe ständig ändert und man damit praktisch keine Gelegenheit hat, das Gadget z.B. einzurahmen. Man müßte praktisch jedesmal nach einer Größenänderung auch den Rahmen neu setzen. Natürlich hat Intuition auch hierfür eine Lösung: Durch Setzen der letzten 4 Bit kann man die entsprechende Seite des Gadgets der Größe automatisch anpassen, ohne sich darum kümmern zu müssen - eine praktische Sache, nicht wahr?

Bit	Wert	Name	Bedeutung
8	256	TOGGLESELECT	Der Zustand des Gadgets wird bei jedem Anklicken umgekehrt, aus Aktiv wird Inaktiv und umgekehrt
9	512	STRINGCENTER	
10	1024	STRINGRIGHT	

Diese beiden Bits legen fest, wie bei Stringgadgets der Text im Eingabefeld erscheinen soll: während ihn ein gesetztes Bit 9 mittig erscheinen läßt, wird er durch Bit 10 rechtsbündig ausgegeben. Ist keines der Bits gesetzt, wird er linksbündig ausgegeben.

Bit	Wert	Name	Bedeutung
11	2048	LONGINT	Macht aus einem String- ein Integer-Gadget
12	4096	ALTKEYMAP	Erlaubt bei der Eingabe eines Stringgadets eine alternative Tastaturbelegung
13	8192	BOOLEXTEND	Zeigt Erweiterung des Boolean-Gadgets durch eine zusätzliche Struktur BOOLINFO (s.u.) an.

Durch das nun folgende Wort enthält den Gadget-Typ:

Bit	Wert	Name	Bedeutung
0	1	BOOLGADGET	Boolean-Gadget
0+1	3	PROPGADGET	Proportional-Gadget
2	4	STRGADGET	String-Gadget

Durch die Bits 4 bis 15 können Sie ein Gadget auch als Systemgadget deklarieren, so z.B. durch Setzen des Bit 4 als Größenveränderungsgadget. Da man diese Gadgets jedoch viel bequemer beim Öffnen des Windows in der NewWindow-Struktur erhalten kann und eine Modifikation meiner Meinung nach überhaupt nicht nötig ist, werde ich Ihnen die Erklärung dieser Bits ersparen.

Das folgende Langwort habe ich schon angesprochen: Es enthält einen Zeiger auf die Border- oder Image-Struktur, mit der wir unser Gadget sichtbar machen können. Falls man es unsichtbar lassen möchte, trägt man hier eine Null ein. Dieser Weg ist nicht immer unsinnig, wenn man z.B. einen Text in das Gadget als Orientierungshilfe einträgt. In

unserem zweiten Beispielprogramm »Chart-Grafik« ist so das Hauptmenü durch zwei Boolean-Gadgets dargestellt, die zwar einen Menütext, jedoch keine Grafik enthalten.

Das folgende Langwort enthält einen weiteren Zeiger, der jedoch nur bei gesetztem GADGHIMAGE-Flag sinnvoll zur Anwendung kommt. Dann muß man nämlich hier die Adresse des Borders oder des Images eintragen, der bei aktiviertem Gadget dargestellt werden soll. Sonst trägt man eine Null ein.

Es folgt ein weiterer Zeiger, diesmal auf eine Textstruktur, deren Texte bei oder in dem Gadget ausgegeben werden sollen. Falls der Text außerhalb des Gadgets liegen soll, sind negative Koordinaten erforderlich, wenn er sich links bzw. oberhalb des Gadgets befinden soll. Die Textstruktur darf wie auch bei den Menüs nicht mit dem Text an sich verwechselt werden. Vielmehr ist der Text ein Bestandteil der Struktur, die aber enthält noch weitere Informationen über Koordinaten, Farben, etc. Ausführlich kommen wir auf sie in Kapitel 2.6. zusprechen.

Das nächste Langwort soll diejenigen Gadgets enthalten, die bei Aktivierung des aktuellen Gadgets deaktiviert werden sollen. Leider funktioniert das aber nicht so recht, so daß man eine Null eintragen muß.

Ein weiteres Langwort muß einen Zeiger auf das sogenannte Spezial-Info enthalten. Bei diesem handelt es sich um eine Struktur, die nur bei Proportional- und Stringgadgets verwendet wird (natürlich damit auch bei Integergadgets) und dazu dient, deren spezielle Eigenschaften zu definieren. Wir kommen auf sie gleich anschließend zurück. Bei Boolean-Gadgets wird eine Null eingetragen.

Als vorletztes haben wir ein Wort, welches die sogenannte ID-Nummer des Gadgets enthält. Diese ist dann relevant, wenn mehrere Gadgets vorhanden sind, um bei der Auswertung des Message-Ports ein Gadget eindeutig identifizieren zu können. Man kann hier eine beliebige Zahl eintragen, muß sich diese aber natürlich für die Auswertung merken.

Zum Schluß folgt noch ein Langwort, das mit einem beliebigen Wert belegt werden darf, ich trage immer eine Null ein, um auch im Listing optisch zu verdeutlichen, daß der Inhalt ohne Bedeutung ist.

2.4.1 Die Bool-Info-Struktur

Diese sehr einfache Struktur enthält drei Einträge, von denen zwei jedoch immer feste Werte erhalten:

```
BOOLINF          macro
dc.w 1          ;immer eins
dc.l \1         ;Zeiger auf Image
dc.w 0          ;immer null
endm
```

Der einzige variable Eintrag stellt den Zeiger auf das Image dar, welches bei Anwahl des Gadgets erscheinen soll.

2.4.2 Die String-Info-Struktur

Die String-Gadgets sind ein Segen für Programmierer, besonders für die, die sich unlängst mit dem C64 herumgequält haben. Man erinnere sich, wie kompliziert Eingaben dort zu programmieren waren. So erschienen im 64'er-Magazin im Laufe der Jahre über zehn Input-Routinen, die eine mehr oder weniger komfortable Eingabe zuließen. Als Besitzer eines Amiga können wir über solche Zustände nur lachen, denn wir haben ja unsere String-gadets.

Die Stringgadets erlauben die Eingabe und/oder Änderung eines Textes, der mit Hilfe verschiedener Funktionen zudem sehr komfortabel editiert werden kann. Wenn man sich vertippt hat, kann man durch eine Undo-Funktion den alten Inhalt des Eingabefeldes wiederherstellen. Zudem kann man die Anzahl der Zeichen festlegen, die maximal eingegeben werden darf. Als besonderen Leckerbissen empfinde ich das Scrollen des Textes innerhalb des Feldes, wenn dieses schmaler ist als der für die Eingabe erforderliche Text.

Nun zu den Einzelheiten: Wenn Sie in der Gadget-Struktur die Höhe und Breite des Gadgets festlegen, wird hierdurch nicht nur mehr der Bereich gekennzeichnet, in dem das Gadget aktiviert werden kann, sondern auch der Platz, der für die Texteingabe zur Verfügung steht. Falls Sie die maximal zulässige Textbreite in der Info-Struktur (siehe unten) größer gewählt haben, als durch die Gadget-Ausmaße zur Verfügung steht, wird der Text gescrollt. Es ist deshalb immer sinnvoll, das Stringgadet durch einen Rahmen (Border) sichtbar zu machen. Um den Text zu editieren, stehen nachfolgende Möglichkeiten zur Verfügung.

Man kann die Eingabe auch beenden, indem man mit der Maus einfach einen Punkt außerhalb des Gadgets anklickt. Es ist wichtig zu beachten, daß bei der Eingabe eines Zeichens mitten im Text das aktuelle Zeichen unter dem Cursor nicht überschrieben wird, sondern daß der Reststring um eine Position nach rechts verschoben wird, d.h., es ist der Insert-Modus aktiv. Diesen kann man auch nicht abschalten; wenn man daher ein Zeichen überschreiben möchte, muß man es mit DEL oder Backspace zunächst löschen und dann das neue Zeichen eintragen.

Taste	**Funktion**
→	Cursor nach rechts
←	Cursor nach links
Shift →	Cursor auf Textende
Shift ←	Cursor auf Textbeginn
Del	Zeichen unter Cursor löschen
Backspace	Zeichen links vom Cursor löschen
Amiga-Q	Macht Änderung rückgängig (mit Undo-Puffer (s.u.)
Amiga-X	Löscht kompletten Eingabepuffer
Return	Beendet Eingabe

Nun möchte ich Sie jedoch nicht weiter auf die Folter spannen, sondern Ihnen unser Makro für die String-Info-Struktur vorstellen:

```
* Struktur für Eingaben bei Stringgadgets
STRINF          macro
dc.l  \1        ;Textpuffer
dc.l  undo      ;Undo-Puffer
dc.w  0         ;Cursor-Position
dc.w  15        ;Anzahl Zeichen (max)
dc.w  0         ;Ausgabeposition Cursor
dc.w  0         ;Position Undo-Puffer
dc.w  0         ;Anzahl Zeichen im Textpuffer
dc.w  0         ;Anzahl sichtbare Zeichen
dc.w  0         ;horizontaler Offset
dc.w  0         ;vertikaler Offset
dc.l  0         ;Zeiger auf Rastport
dc.l  0         ;Integer-Wert
dc.l  0         ;Tastatur-Belegung
endm
```

Das erste Langwort stellt den Zeiger auf den Textpuffer dar. Dieser muß so groß bemessen sein, daß er die maximal zulässige Anzahl der Zeichen (siehe unten) aufnehmen kann. Es folgt ein weiterer Zeiger, diesmal auf den sogenannten Undo-Puffer. In diesen wird bei jedem Aufruf des Gadgtes der aktuelle Inhalt des Textpuffers übertragen. Wenn man diesen modifiziert hat und anschließend die Tastenkombination Amiga-Q betätigt, wird der Inhalt des Undo-Puffers in den Textpuffer zurückgeschrieben, so daß man die gleiche Ausgangslage hat wie vor der Eingabe. Da man immer nur ein Gadget gleichzeitig bearbeiten kann, braucht man natürlich für ein Window auch nur einen Undo-Puffer. Dieser muß so bemessen sein, daß der Platz auch für das Gadget mit der größten Anzahl von Zeichen ausreicht.

Es folgt die Start-Cursorposition innerhalb des Textes. Wenn man wie üblich den Cursor auf dem ersten Zeichen positionieren möchte, muß man eine Null eintragen, da Computer ja anders als die Menschen üblicherweise nicht mit eins, sondern mit null zu zählen beginnen. Das nächste Wort enthält die maximal zulässige Anzahl der Zeichen. Falls man versucht, mehr Zeichen einzugeben, blinkt der Bildschirm auf, um anzuzeigen, daß jeder weitere Eingabeversuch zwecklos ist. Man darf diese Zeichenzahl nicht mit der darstellbaren Zeichenzahl verwechseln, die geringer sein darf. Es wird dann gescrollt.

Im folgenden Wort kann man festlegen, ab welcher Position des Textpuffers der Text im Eingabefeld ausgegeben wird. Wenn man den Anfang des Textes sehen möchte, muß man auch hier eine Null eintragen. Die folgenden fünf Wörter und das darauffolgende Langwort werden von Intuition beschrieben und sind für den Anwender normalerweise von geringer Bedeutung. Bei der Initialisierung kann man beliebige Werte hineinschreiben, wir setzten Nullen ein.

Das nächste Langwort wird ebenfalls von Intuition beschrieben, ist aber ungleich wichtiger als sein Vorgänger. Hier wird nämlich bei der Deklaration eines Integergadgets der Integerwert des Ergebnisses eingetragen. Das letzte Langwort muß bei Verwendung des ALTKEYMAP-Flags den Zeiger auf die alternative Tastaturtabelle enthalten, wenn man die Standardtastaturbelegung benutzen möchte, trägt man eine Null ein, obwohl der Wert irrelevant ist, sobald das ALTKEYMAP-Flag nicht gesetzt wurde. Damit sind wir schon am Ende der String-Gadgets angekommen und wenden uns nun den Proportionalgadgets zu.

2.4.3 Die Prop-Info-Struktur

Um ein Proportionalgadget aufbauen zu können, muß die nun folgende Special-Info-Struktur aufgebaut werden:

```
PROPINFO macro
dc.w \1         ;Flags
dc.w 0,0        ;x,y-Position
dc.w \2         ;Schrittweite
dc.w 0          ;keine vertikale Verschiebung
dc.w 0          ;Breite
dc.w 0          ;Höhe
dc.w 0          ;Schrittweite horizontal
dc.w 0          ;Schrittweite vertikal
dc.w 0          ;linker Rand
dc.w 0          ;rechter Rand
endm;
```

Das erste Wort enthält fünf Bit-Flags. Am interessantesten ist dabei Bit 0 (AUTOKNOB). Durch dieses Bit kann man festlegen, ob die Grafikdaten für den Schieberegler von dem Programmierer oder von Intuition gesetzt werden sollen. Ist das Bit gelöscht, bedeutet es, daß man seinen eigenen Schieberegler definieren möchte. Dazu muß man in das zweite Langwort der Gadget-Struktur (siehe oben) einen Zeiger auf das Image für den Regler eintragen. Ist das Bit jedoch gesetzt, wird der Regler automatisch von Intuition generiert. Hierbei handelt es sich um einen einfachen Balken, wie Sie ihn bestimmt schon einmal in professionellen Programmen gesehen haben. Wenn nicht, lassen Sie sich überraschen. In diesem Fall muß in das Langwort der Gadget-Struktur nicht etwa gar nichts eingetragen werden, wie man das erwarten könnte, sondern ein Zeiger auf eine kleine, weitere Struktur, die allerdings komplett von Intuition beschrieben wird:

```
dc.w  0         ;x-Position des Schiebers in der Box
dc.w  0         ;y-Position des Schiebers in der Box
dc.w  0         ;Breite des Reglers
dc.w  0         ;Höhe des Reglers
```

Da man diese Werte nicht zu initialisieren braucht, muß man nur vier Wörter an Speicherplatz reservieren. Die folgenden Bits legen fest, in welche Richtung der Regler verschoben werden kann und ob er umrahmt dargestellt wird:

Bit	Wert	Name	Bedeutung
1	2	FREEHORIZ	Horizontale Verschiebung möglich
2	4	FREEVERT	Vertikale Verschiebung möglich
3	8	PROPBORDERLESS	Keine automatische Umrahmung

In die nächsten beiden Worte wird die x- sowie y-Position des Reglers beim Aufruf des Gadgets eingetragen. Die Werte können von 0 bis $FFFF reichen, wobei 0 den linken bzw. oberen Rand bedeutet und $FFFF den rechten bzw. unteren Rand. Um also z.B. den Schieber genau mittig zu positionieren, gibt man eine $8000 ein.

Das nächste Wort hat auf zweierlei Weise Bedeutung. Zum einen legt es die Schrittweite des Reglers fest, wenn man diesen nicht kontinuierlich mit der Maus verschiebt, sondern einfach die Maus anklickt. Die Schrittweite wird im Verhältnis zu der Gesamtbreite der Reglerbox angegeben: Möchten Sie sie z.B. auf $^1/_{10}$ der Gesamtbreite des Gadgets festlegen, müssen Sie eine $FFFF/10 eintragen. Es ist klar, daß keine beliebigen Verhältnisse, sondern nur Integerwerte erreicht werden können. Bei Verwendung des AUTOKNOBS wird durch dieses Wort zusätzlich dessen Breite festgelegt. Im obigen Beispiel würde der Schieber daher $^1/_{10}$mal so breit sein wie der gesamte Reglerbereich.

Die folgenden sechs Wörter werden von Intuition beschrieben und brauchen daher nicht initialisiert zu werden. Damit sind wir, liebe Leser, schon am Ende des Abschnitts über Gadgets angekommen! Bislang haben wir viel über die Programmiertechniken von Intuition kennengelernt. Den wichtigsten Abschnitt haben wir aber noch nicht behandelt, nämlich wie wir die Informationen auswerten und damit für uns nutzbar machen können. Dazu dient der Intuition-Message-Port, den wir nun besprechen werden.

2.5 Der Intuition-Message-Port

In dem Abschnitt über Windows habe ich Ihnen erklärt, wie wir unseren Amiga durch Setzen diverser IDCMP-Flags dazu bringen können, uns Mitteilungen über diese Ereignisse zukommen zu lassen. Da eine ausführliche Beschreibung dieser Flags dort jedoch noch zu früh war und nicht ins Konzept paßte, habe ich sie mir bis zum jetzigen Zeitpunkt aufgehoben. Man kann die wichtigsten dieser Flags in sieben Gruppen einteilen:

1) Die Mouse-Flags
2) Die Menü-Flags
3) Die Gadget-Flags
4) Die Requester-Flags
5) Die Window-Flags
6) Die Tastatur-Flags
7) Sonstige Flags

2.5.1 Die Mouse-Flags

MOUSEBOTTONS	Dieses Flag signalisiert, daß eine Maustaste gedrückt oder losgelassen wurde. Sinnvoll ist dies nur bei der linken Taste, da ein Druck der rechten Maustaste normalerweise über Menupick gemeldet wird. Eine Ausnahme besteht im Anklicken eines Gadgets. Dieses Ereignis wird über die Gadget-Flags gemeldet.
MOUSEMOVE	Hierbei werden die x/y-Koordinaten bei jeder Mausbewegung gemeldet. Voraussetzung ist aber das Setzen des Window-Flags REPORTMOUSE (Bitte verwechseln Sie diese Flags niemals mit den IDCMP-Flags!) Wenn Sie die Koordinaten auch von Gadgets erfahren wollen, muß zusätzlich in der Gadget-Struktur das Flag FOLLOWMOUSE gesetzt werden.
DELTAMOUSE	Dieses Flag kann nur zusammen mit MOUSEMOVE gesetzt werden und bewirkt, daß die Koordinaten nicht mehr absolut, sondern relativ zum letzten Standort der Maus gemeldet werden. Die Benutzung dieses Flags ist aber weniger empfehlenswert, da die relativen Koordinaten auch noch dann gemeldet werden, wenn sich die Maus außerhalb des Windows befindet!

2.5.2 Die Menü-Flags

MENUPICK	Bei Setzen dieses Flags werden Sie informiert, sobald der Anwender die rechte Maustaste betätigt hat. Den Code für den angewählten Menüpunkt können Sie in der Adresse $18 des Message-Ports (siehe unten) ablesen. Falls gar kein Punkt angewählt wurde, sondern die Maustaste einfach wieder losgelassen wurde, steht hier der Wert $FFFF./
MENUVERIFY	Bei diesem Flag werden Sie darüber informiert, daß der Anwender ein Menü wählen möchte. Dieses klappt jedoch erst dann herunter, wenn Sie die Message mit REPLY beantworten (siehe unten). Im allgemeinen ist der Einsatz dieses Flags nicht notwendig, da Sie ja auch, wenn ein Menüpunkt heruntergeklappt wurde, auf die Anwahl eines Punktes nicht zu reagieren brauchen. Wenn Sie überhaupt kein Menü brauchen können, schalten Sie es einfach ab, wie wir es im Abschnitt über die Menüs praktiziert haben.

2.5.3 Die Gadget-Flags

GADGETDOWN	Falls Sie dieses Flag zusammen mit dem Gadget-Flag GADGIMMEDIATE gesetzt haben, werden Sie sofort informiert, wenn der Anwender über dem Gadget-Feld die linke Maustaste gedrückt hat.
GADGETUP	Im Gegensatz zum letzten IDCMP-Flag wird hier bei zusätzlich gesetztem Gadget-Flag RELVERIFY nur dann gemeldet, wenn die linke Maustaste über dem Gadget betätigt und dann auch wieder losgelassen wird. Nur bei dieser Betriebsart kann man wirklich sicher sein, daß das Gadget wirklich angewählt werden soll, weshalb sie GADGETDOWN vorzuziehen ist.

2.5.4 Die Requester-Flags

REQSET	Bei gesetztem Flag werden Sie darüber informiert, wann von mehreren möglichen Requestern deren erster in dem Window erscheint.
REQCLEAR	Analog bekommen Sie hier die Meldung, wenn der letzte Requester wieder verschwunden ist.
REQVERIFY	Dieses Flag entspricht dem MENUVERIFY bezogen auf Requester. Sie werden also informiert, bevor der Requester auf dem Bildschirm erscheint. Bei mehrereren Requestern wirkt dieses Flag nur auf den ersten.

2.5.5 Die Window-Flags

CLOSEWINDOW	Dieses Flag müssen Sie setzen, wenn Sie dem Window ein Close-System-Gadget spendiert haben. Sie werden nur dann informiert, wenn der Anwender dieses angeklickt hat.

NEWSIZE	Das Setzen dieses Flags führt dazu, daß Sie eine Nachricht erhalten, sobald die Windowgröße verändert wurde. Die neuen Daten kann man dann aus der Window-Struktur ablesen.
REFRESHWINDOW	Hierbei werden Sie informiert, wenn Ihr Window ein Refreshing (Neuzeichnen) benötigt. Das Flag braucht natürlich nur bei Windows vom Typ SIMPLE_REFRESH oder SMART_ REFRESH eingesetzt zu werden, wobei bei letzterem Typ nur der Fall von Interesse ist, wenn das Window vergrößert und anschließend wieder verkleinert wird. Völlig sinnlos ist der Einsatz bei Windows, bei denen überhaupt keine Größenänderung zugelassen ist.
SIZEVERIFY	Durch Setzen dieses Flags können Sie den Anwender im eigenen Saft schmoren lassen, wenn er die Windowgröße verändern möchte. Sie bekommen nämlich schon dann Nachricht, wenn die Aktion an sich noch nicht ausgeführt wurde. Sie wird auch solange nicht durchgeführt, bis die Message mit REPLY beantwortet wird. Falls Sie z.B. keine Lust auf Refreshing haben, lassen Sie Ihr Programm solange in einer Schleife verharren, bis der Benutzer entnervt aufgibt.
ACTIVEWINDOW	Gesetztes Bit führt zur Meldung, wenn Window aktiviert wurde.
INACTIVEWINDOW	Funktioniert analog bei Inaktivierung des Windows.

2.5.6 Die Tastatur-Flags

RAWKEY	Falls Sie dieses Flag setzen, können Sie anschließend an der Adresse $18 des Message-Ports den Scan-Code der jeweils gedrückten Taste auslesen. An der Adresse $1A findet man Zusatzinformationen, z.B. ob die CTRL- oder SHIFT-Taste gedrückt wurde. Da der Amiga jedoch ein Maus-Computer ist und Eingaben besser über Gadgets als über die Tastatur erfolgen, sollten Sie immer dann, wenn Sie dieses Flag benutzen, einmal über eine mögliche Maus-Alternative nachdenken.

VANILLAKEY	Im Gegensatz zu RAWKEY erhalten Sie hier den Tastencode an der Adresse $18 des Ports im ASCII-Code gemäß der durch SetMap festgelegten Tabelle. Diese Funktion ist im allgemeinen vorzuziehen, versagt aber z.B. bei der Funktionstastenabfrage, so daß man hier auf RAWKEY zurückgreifen muß.

2.5.7 Sonstige Flags

NEWPREFS	Durch Setzen dieses Flags werden Sie darüber informiert, wenn Änderungen in der Preferences-Struktur vorgenommen werden. Falls Sie diese in Ihrem Programm auslesen, ist eine Information unbedingt nötig. Wer, werden Sie nun einwenden, kann denn in meinem eigenen Programm ohne mein Wissen diese Struktur ändern? Nun, dort natürlich niemand. Andere Tasks sind dazu jedoch in der Lage, während Ihr Programm sanft vor sich hinschlummert.
DISKINSERTED	Bei gesetztem Flag erhalten Sie Nachricht, wenn eine Diskette ins Laufwerk eingelegt wird. Sie können den Anwender damit von zusätzlicher Arbeit in Form des Anklicken eines Requesters entlasten.
DISKREMOVED	Meldet analog, wenn Diskette aus Laufwerk entfernt wird.

2.5.8 Der Intuition-Message-Port

Jetzt sind wir an dem entscheidenen Punkt angelangt, der Übermittlung von Nachrichten Intuitions an das Programm. Dazu gibt es eine Funktion in der Exec-Library. Bevor diese aufgerufen werden kann, muß jedoch zunächst das Register A0 mit der Adresse des User (=Message)-Port des Windows geladen werden, den man ab der Adresse $56 in der Window-Struktur findet:

```
move.l windowpointer,a0          ;Zeiger auf Window-Struktur
move.l $56(a0),a0                ;User-Port holen
CALLEXEC GetMsg                  ;Message holen
```

Im Register D0 wird dann ein Wert zurückgegeben, der ein Zeiger auf den Intuition-Message-Port darstellt. In diesem steht u.a. an der Adresse $14 ein Wert, der haargenau dem des entsprechenden IDCMP-Flags des auslösenden Ereignisses entspricht, so z.B. eine $200, falls das Close-Gadget betätigt wurde. Was aber passiert, wenn überhaupt kein

Ereignis stattfand? Dann wird eine Null zurückgegeben. Das Problem besteht nun darin, daß man normalerweise in einer Schleife solange warten müßte, bis ein Ereignis eingetreten ist und damit ein Wert ungleich Null übergeben wird (Polling). Diese Technik hat jedoch den Nachteil, daß der Prozessor ständig beschäftigt ist. Damit wird eventuellen anderen Tasks Zeit gestohlen, was sich in deutlich verlangsamter Programmausführung bemerkbar machen kann. Da diese weiteren Tasks jedoch mehr theoretisch zur Anwendung kommen (im allgemeinen jedenfalls), kann man das Polling nicht eindeutig als Programmierfehler kennzeichnen. Die entsprechende Routine sieht so aus:

```
wait    move.l windowpointer,a0          ;Zeiger auf Window-Struktur
        move.l $56(a0),a0                ;User-Port holen
        CALLEXEC GetMsg                  ;Message holen
        tst.l d0                         ;warten, bis ungleich Null
        beq wait
```

Die weitaus elegantere Möglichkeit besteht jedoch darin, den eigenen Task bis zum Eintreffen einer Message »schlafen« zu legen. Dafür existiert eine weitere Exec-Routine, die neben der Adresse des User-Ports auch noch ein sogenanntes Signal-Bit benötigt. Dieses findet sich in der Adresse $0F des User-Ports und muß im Register D0 übergeben werden. Nachdem man die Message erhalten hat, muß man diese mit einer weiteren Funktion quittieren:

```
wait    move.l windowpointer,a0
        move.l 86(a0),a0                ;User-Port holen
        move.l a0,a5                    ;Adresse retten
        move.b 15(a0),d1                ;Signal-Bit holen
        moveq #0,d0                     ;Nummer in Maske
        bset d1,d0                      ;wandeln
        CALLEXEC Wait                   ;Auf Wiedersehen!
        move.l a5,a0
        CALLEXEC GetMsg                 ;Message holen
        move.l d0,d7                    ;und retten
        move.l d0,a1                    ;für Reply
        CALLEXEC Reply                  ;Message quittieren
```

Dieser Weg ist natürlich dem Polling vorzuziehen, da der Prozessor nicht mit einer Warteschleife belastet wird. Da aber beide Wege funktionieren, habe ich auch beide in den Beispielprogrammen verwendet.

Kritisch betrachtet, läßt die Message aber noch viele Wünsche offen: Gut, wir wissen, daß ein Menüpunkt ausgewählt wurde oder daß ein Gadget angeklickt wurde. Um welchen Menüpunkt oder um welches Gadget es sich handelt, bleibt aber bislang unbekannt. Hierfür müssen wir noch ein bißchen mehr auswerten.

2.5.9 Die Auswertung von Menüs

Wie wir aus dem Abschnitt über die Menüs wissen, wird uns der angewählte Menüpunkt codiert mitgeteilt. Dieses Langwort steht ab der Adresse $18 des Message-Ports. Durch geschickte Bit-Schiebung können wir die Nummern des Menüs, Menüpunktes und Untermenüpunktes herausanalysieren:

```
wait    move.l windowpointer,a0
        move.l 86(a0),a0                ;User-Port holen
        move.l a0,a5                    ;Adresse retten
        move.b 15(a0),d1                ;Signal-Bit holen
        moveq #0,d0                     ;Nummer in Maske
        bset d1,d0                      ;wandeln
        CALLEXEC Wait                   ;Auf Wiedersehen!
        move.l a5,a0
        CALLEXEC GetMsg                 ;Message holen
        move.l d0,a1                    ;für Reply
        move.w $18(a1),d7               ;Menüpunkt holen
        CALLEXEC ReplyMsg               ;Message quittieren
        cmp.w #$ffff,d7                 ;kein Menüpunkt gewählt
        beq wait
        move.w d7,d6
        lsr #7,d7
        lsr #4,d7
        lsl #2,d7                       ;Untermenü*4 in D7
        move.w d6,d5
        and.l #%00000000000000000000011111100000,d6
        lsr #3,d6                       ;Menüpunkt*4 in D6
        and.l #%00000000000000000000000000011111,d5
        lsl #2,d5                       ;Menü*4 in D5
```

Zunächst wird überprüft, ob ein Menüpunkt angewählt wurde, oder ob die rechte Maustaste vorher wieder losgelassen wurde, was aus dem Wert $FFFF für den Menücode ersichtlich wäre. Wenn dies der Fall ist, wird auf die nächste Message gewartet. Dann wird ausgewertet: Durch elfmaliges Nach-RechtsrSchieben wird die Untermenünummer herauskristallisiert. Da wir diese, wie auch die übrigen Daten, jedoch als Tabellenzeiger benutzen wollen (Langwörter), muß die Nummer durch zweimaliges Nach-Links-Schieben wieder mal 4 genommen werden. Der Menüpunkt wird gefunden, indem man die Bits 5–10 ausblendet und das Ergebnis dreimal nach rechts verschiebt. Damit haben wir auch die Menüpunktnummer mal 4 gefunden. Analog wird mit der Menünummer verfahren. Jetzt kommt der interessante Teil, der die eigentliche Auswertung vornimmt. Er ist so universell gehalten, daß Sie ihn in allen eigenen Programmen verwenden können, egal, wie viele Menüs mit wie vielen Menüpunkten Sie verwenden! Dies ist der kleine Unterschied zu den Routinen aus anderen Büchern, die zwar ihre Demomenüs optimal auswerten, bei kleinsten Veränderungen jedoch schon versagen.

```
        move.l #tab,d4
        add.l d5,d4
        move.l d4,a4
        move.l (a4),d4                  ;Menüpunkt feststellen
        add.l d6,d4
        move.l d4,a4
        move.l (a4),a4
        jmp (a4)                        ;und anspringen

tabmenu1      dc.l item11,item12,...,item1na  ;Menüpunkte 1-na Menü1
tabmenu2      dc.l item21,item22,...,item2nb  ;Menüpunkte 1-nb Menü2
........         ..........................
tabmenum      dc.l itemm1,itemm2,...,itemmnx  ;Menüpunkte 1-nx Menüm
tab           dc.l tabmenu1,tabmenu2,...,tabmenum ;Menüs 1 bis m
```

Das einfache, aber wirkungsvolle Prinzip funktioniert wie folgt: In einer Haupttabelle (tab) ist jedem Menü ein Tabellenwert zugeordnet, der auf eine weitere Untertabelle (tabmenu) zeigt. Daher existieren soviele Untertabellen, wie Menüs vorhanden sind. Programmtechnisch wird zunächst die Startadresse in das Register D4 geladen und dazu die Menünummer 4 addiert. Damit entsteht ein Zeiger auf das Langwort, welches der Menünummer entspricht. Dieser Wert wird dann in das Register A4 übertragen und anschließend der Inhalt, nämlich die Adresse der zugehörigen Untertabelle, geholt. Die Untertabellen ihrerseits enthalten die Adressen der Programmroutinen der einzelnen Menüpunkte. Diese werden auf die gleiche Weise wie eben beschrieben geholt und zu guter Letzt mit einem indirekten Sprungbefehl angesprungen. Die Label *item11* bis *itemmn* sind also nichts anderes als Zeiger auf die den Menüpunkten zugehörigen Routinen, die dann ausgeführt werden. Die Subitems konnten auf diese Weise aber nicht erfaßt werden: für sie wurde eine neue Auswertung in der Routine des entsprechenden Menüpunktes vorgenommen. Wir wollen einmal annehmen, daß der Menüpunkt 2 des ersten Menüs Untermenüpunkte enthalten soll:

```
item12  move.l #subtab,d4
        add.l d7,d4
        move.l d4,a4
        move.l (a4),a4                  ;Untermenüpunkt feststellen
        jmp (a4)

subtab  dc.l subitem1,subitem2,...,subitemn
```

Sie sehen, es wird nach der gleichen Methode gearbeitet. Die endgültigen Einsprungadressen werden durch die Label *subitem1* bis *subitemn* dargestellt, wenn n Subitems existieren. Diese Routine müssen Sie natürlich nur bei solchen Menüpunkten installieren, die auch Untermenüpunkte besitzen. Sonst wäre hier item12 schon die endgültige Adresse. Es ist erstaunlich, mit wie wenig Aufwand man eine universelle Routine programmieren kann, finden Sie nicht auch?

2.5.10 Die Auswertung von Gadgets

Die entscheidende Frage nach einer Gadget-Message ist die, welches Gadget angewählt wurde (wenn nur ein Gadget existiert, besteht diese Problematik natürlich nicht). Nun, dazu existiert ein weiterer Zeiger im Message-Port an der Adresse $1C. In ihm steht die Adresse der Struktur des jeweils angewählten Objektes, also auch des Gadgets. Damit ist das Problem bereits gelöst: Da wir in unserer Gadget-Struktur ja u.a. die ID-Nummer an der Adresse $26 abgelegt haben (rechnen Sie ruhig einmal nach), können wir uns diese leicht holen und damit auf die Message reagieren. Die beschriebenen Vorgänge werden von folgender Routine erledigt, die Adresse des Message-Portes nach CALLEXEC GetMsg soll immer noch im Register D0 vorhanden sein:

```
move.l d0,a2
move.l $1C(a2),a2       ;Adresse Gadget aus Message-Port
move.w $26(a2),d0       ;ID-Nummer aus Gadget-Struktur holen
```

Es ist eminent wichtig zu beachten, welche Pointer Langwörter, Wörter oder Bytes sind! So stellt der Zeiger auf die Gadget-Struktur ein Langwort dar, die ID-Nummer aber nur ein Wort! Hätte man hier auch versucht, ein Langwort zu lesen, hätte man natürlich einen völlig falschen Wert erhalten und das Programm könnte nicht reagieren, vielleicht wäre sogar der Guru wieder zu seinem Recht gekommen!

2.5.11 Die Auswertung der Tastatur

Diese Auswertung ist relativ einfach. Der Code wird im selben Feld zurückgegeben, wie auch die Menünummer. Daher genügt ein Zweizeiler, wenn man die Startadresse des Message-Ports geholt hat:

```
move.l d0,a2
move.l $18(a2),a2       ;Tastaturcode aus Message-Port
```

2.5.12 Die Auswertung der übrigen Events

Die übrigen Events haben eins gemeinsam: Sie teilen uns nur mit, daß irgend etwas passiert ist, diese Sache ist aber eindeutig. So wird mitgeteilt, daß eine Diskette eingelegt wurde, welche, kann Intuition aber nicht wissen. Genauso wird mitgeteilt, daß das Fenster vergrößert wurde, die neuen Daten müssen aber aus der Window-Struktur gelesen werden.

Deshalb reicht es bei diesen Events, nur das IDCMP-Flag aus dem Message-Port zu lesen. Man kann auf Grund dieser Ausgabe eine eindeutige Schlußfolgerung ziehen, anders als bei den Menüs, den Gadgets und den Tastatureingaben. Wie man an den IDCMP-Code kommt, habe ich oben schon beschrieben:

```
move.l d0,a2
move.l $14(a2),a2       ;IDCMP-Flag aus Message-Port
```

Falls man mehrere Events gleichzeitig abfragen muß, ist es sowieso nötig, zunächst an Hand des IDCMP-Flags festzustellen, was für ein Ereignis überhaupt vorliegt. Erst dann kann man in die einzelnen Auswertungs-Routinen für Menüs etc. springen.

2.6 Die Textstruktur

Im Abschnitt über die Menüs und Gadgets habe ich eine sogenannte Textstruktur erwähnt. In dieser sehr kurzen Struktur kann man noch einige Eigenschaften des Textes definieren, den man ausgeben möchte:

```
TEXTOUT          macro
                 dc.b 3,0        ;Farben
                 dc.b 0          ;Farbmodus
                 even
                 dc.w  0         ;x-Koordinate
                 dc.w  0         ;y-Koordinate
                 dc.l  0         ;Zeichensatz
                 dc.l \1         ;Textzeiger
                 dc.l  0         ;kein weiterer Text
                 endm
```

Die ersten beiden Bytes sind für die Farben zuständig. Während man mit der Vordergrundfarbe das Aussehen des Textes tatsächlich beeinflussen kann, bleibt die Hintergrundfarbe ohne Wirkung, da sie ja global für das gesamte Window definiert wurde. Mit dem folgenden Byte kann man zwischen normaler Darstellung (0) und invertierter Darstellung (4) wählen. Nachdem die Adresse begradigt wurde, folgen zwei Byte für die Koordinaten. Diese sind an sich überflüssig, da man diese ja schon in der Menü- bzw. Gadget-Struktur festgelegt hat. Als nächstes haben wir ein Langwort, das den Zeiger auf den eigentlichen Text darstellt. Dieser muß mit einem Nullbyte abgeschlossen werden.

Der letzte Eintrag stellt einen Zeiger auf eine etwaige noch folgende Textstruktur dar. Man kann so mit einem Aufruf beliebig viele Texte verketten. Damit sind wir schon am Ende angekommen. Sie sehen, der Name Textstruktur ist im Vergleich zu dem, was wir bislang behandelt haben, fast schon überheblich.

2.7 Textausgabe in ein Window

Die Textausgabe ist sehr einfach, da für sie eine Funktion zur Verfügung steht. Diese wird wie folgt aufgerufen:

```
                move.l windowpointer,a0    ;Rast-Port aus
                move.l 50(a0),a0           ;Window-Struktur holen
                move.l #text,a1            ;Zeiger auf Textstruktur
                move.l #xkoord,d0          ;x-Koordinate
                move.l #ykoord,d1          ;y-Koordinate
                CALLINT PrintIText         ;Text ausgeben
text            TEXTOUT  ausgabetext
ausgabetext     dc.b "Bitte diesen Text drucken!",0
```

Der einzige interessante Punkt ist das Auslesen des Rast-Ports aus der Window-Strukur. Mit den Informationen aus dem Abschnitt 2.2. ist dies aber für uns kein Problem mehr.

2.8 Die Images

Wenn Sie bisher alle Abschnitte gelesen haben, werden Sie festgestellt haben, daß praktisch in jedem Abschnitt über die sogenannten Images die Rede war. Hierbei handelt es sich um rechteckige, bitweise definierte Grafiken, die mit sehr hoher Geschwindigkeit gezeichnet werden können. Beispiele für Images stellen z.B. die Systemgadgtes oder die Diskettensymbole der Workbench dar. Um ein Image zu definieren, muß man die Grafikdaten bitweise in einem 16-Bit-Datenarray ablegen. Dabei bedeutet ein gesetztes Bit, daß der Punkt gesetzt wird, ein nicht gesetztes Bit stellt daher einen gelöschten Punkt dar. Die Größe des Images ist theoretisch unbegrenzt, die Breite in Pixeln muß aber ein vielfaches von 16 sein, damit Sie die Daten zeilenweise anlegen können (in Wörtern).

Problematisch wird die Sache erst dann, wenn mit mehreren Bitplanes gearbeitet wird. Dann reichen die Bit-Daten allein nämlich nicht mehr aus, um die Grafik zu definieren. Deshalb existiert auch für Images eine Struktur, in der wir definieren können, welche Bitplanes wie genutzt werden sollen. Gezeichnet wird ein Image durch Aufrufen einer weiteren Intuition-Funktion mit dem Namen DrawImage. Die zu übergebenden Parameter entsprechen denen der Textausgabe, wobei der Zeiger jetzt natürlich nicht mehr auf die Text-, sondern auf die Image-Struktur zeigt.

```
move.l windowpointer,a0
move.l 50(a0),a0          ;Rastport holen
move.l image,a1           ;Adresse Image-Struktur
move.l #xkoord,d0         ;relative x-Koordinate
move.l #ykoord,d1         ;relative y-Koordinate
CALLINT DrawImage         ;Image zeichnen
```

Die Koordinaten beziehen sich, wie beim Text auch, immer auf die linke obere Ecke des aktuellen Containers, im Regelfall also auf das angeklickte Window.

2.8.1 Die Image-Struktur

Mit neun Einträgen gehört zwar auch die Image-Struktur zu den Zwergen, bei vielen verwendeten Grafiken lohnt es sich aber doch, ein Makro anzulegen:

```
IMAGE           macro
                dc.w  0,0               ;x- und y-Koordinate
                dc.w  \1                ;Breite des Images
                dc.w  \2                ;Höhe des Images
                dc.w  \3                ;Anzahl Bitplanes
                dc.l  \4                ;Zeiger auf Imagedaten
                dc.b  \5                ;Zeichnung (Bitplane)
                dc.b  \6                ;Hintergrund (Bitplane)
                dc.l  0                 ;kein weiteres Image
```

Die ersten beiden Wörter sind belanglos, da wir die Koordinaten schon beim Aufruf festgelegt haben. Dann folgen die Höhen- und Breitenangabe jeweils in Pixel. Wie oben schon ausgeführt wurde, sollte man die Breite immer in Vielfachen von 16 angeben, da die Bits zeilenweise eingelesen werden. Pro Wort lassen sich genau 16 Bit definieren, so daß man z.B. mit zwei Wörtern eine Zeile mit 32 Grafikpunkten definieren kann.

Das folgende Wort legt fest, mit wievielen Bitplanes gearbeitet wird und damit, aus wievielen Farben unser Image maximal bestehen kann. Das nächste Langwort stellt einen Zeiger auf die eigentlichen Image-Daten, das Bitmuster, dar. In diesem werden, wie schon erwähnt, die Bits zeilenweise eingelesen. Was aber passiert, wenn wir z.B. drei Bitplanes haben und ein 16 x 16 Punkt großes Image darstellen wollen? Nun, Intuition liest die Daten Bitplane für Bitplane ein. Um also die erste Bitplane mit Daten zu füllen, sind 16 Wörter notwendig (16 Zeilen mal jeweils einem Wort). Intuition holt sich also die ersten 16 Wörter aus der Struktur. Je nachdem, wie viele Bitplanes wir festgelegt haben, werden nun nochmals jeweils 16 Wörter für die folgenden Bitplanes, hier die Bitplanes zwei und drei, geholt.

Interessant ist aber besonders der Fall, wenn wir mehrere Bitplanes zur Verfügung haben aber nicht in alle, sondern nur in wenige oder gar eine ausgewählte zeichnen möchten. Dafür steht das folgende Byte, PlanePick genannt, zur Verfügung. In diesem Byte wird jede Bitplane durch ein Bit repräsentiert, wobei Bit 0 der Plane 0 entspricht, Bit 1 der Plane 1 usw. Angenommen, wir haben in unserem Window fünf Bitplanes definiert, möchten aber nur zwei von diesen beschreiben, um nicht endlose Bit-Strukturen aufbauen zu müssen. Zunächst müssen wir uns darüber im klaren sein, in welchen Farben unser Image erscheinen soll, da diese ja unmittelbar von den verwendeten Bitplanes abhängt. Aus dem Abschnitt über die Screens wissen wir, daß die an gleicher Position liegenden Bits jeder Bitplane dazu verwendet werden, in ihrer Summe die Farbregisternummer zu bestimmen, aus der die Information für den Punkt geholt wird. Wir wollen unsere Grafik z.B. Rot gestalten. Da sich Rot normalerweise im Register 3 (=21+20) befindet, müssen wir offenbar die Bitplanes 0 und 1 zu Rate ziehen. Daher setzen wir auch im PlanePick-Byte die Bits 0 und 1, d.h., wir tragen den Wert 3 in unsere Image-Struktur ein. Dadurch müssen wir nur noch 32 Wörter statt deren 80 mit Bit-Daten definieren, wenn wir eine 16 x 16 Punkte große Grafik aufbauen möchten.

Das folgende Byte mit dem Namen PlaneOff stellt eine ebenso interessante Funktion zur Verfügung: Auch hier entspricht jedes Bit einer Plane. In den hierdurch festgelegten Planes werden jedoch genau die Bits gesetzt, die in der Image-Struktur gelöscht sind und umgedreht. Man kann also sagen, daß durch dieses Byte die Hintergrundfarbe des Images festgelegt wird. Möchte man z.B. einen weißen Hintergrund, setzt man das Bit 1, um die erste Plane auszuwählen.

Das letzte Langwort schließlich deutet auf eventuell weitere Image-Strukturen. Dadurch ist es möglich, mit nur einem Draw-Image-Befehl mehrere Images auszugeben. Um Ihnen auch einmal den praktischen Aufbau der Bit-Daten zu veranschaulichen, folgen nun sieben

Wörter, die einen Pfeil nach oben darstellen. Da wir nur mit einer Bitplane arbeiten, wird unsere Grafik das Ausmaß von 16 x 7 Punkten annehmen.

```
dc.w %0000001110000000
dc.w %0000011111000000
dc.w %0000111111100000
dc.w %0001111111110000
dc.w %0000001110000000
dc.w %0000001110000000
dc.w %0000001110000000
```

2.9 Die Borders

Bei den Borders handelt es sich nicht, wie der Name vermuten läßt, um eine Funktion nur zum Zeichnen eines einfachen Rahmens. Man kann nämlich beliebig viele Linien zeichnen, die einen beliebigen Winkel zueinander einnehmen dürfen. Die Hauptanwendung wird aber darin liegen, z.B. String- oder Proportionalgadets sichtbar zu machen. Der Aufruf geschieht völlig analog zu den Images:

```
move.l windowpointer,a0
move.l 50(a0),a0          ;Rastport holen
move.l #border,a1         ;Adresse Border-Struktur
move.l #xkoord,d0         ;Startpunkt x-Koordinate
move.l #ykoord,d1         ;Startpunkt y-Koordinate
CALLINT DrawBorder        ;Border zeichnen
```

Weitere Sachen sind eigentlich nicht zu sagen, so daß wir gleich zur Border-Struktur übergehen können.

2.9.1 Die Border-Struktur

Die Border-Struktur ist noch einfacher aufgebaut als die der Images. Sie enthält nur acht Einträge:

```
BORDER          macro
                dc.w 0,0            ;x- und y-Koordinaten
                dc.b 3              ;Zeichenfarbe
                dc.b 0              ;Hintergrundfarbe
                dc.b \1             ;Zeichenmodus
                dc.b \2             ;Anzahl Koordinatenpaare
                dc.l \3             ;Zeiger auf Koordinatentabelle
                dc.l 0              ;keine weitere Struktur
                endm
```

Die ersten beiden Einträge sind wiederum unbedeutend, da wir den Startpunkt unseres Polygons schon beim Aufruf definiert haben. Das folgende Byte gibt die Zeichenfarbe der

Linien an. Die Hintergrundfarbe ist global durch das Window festgelegt, weshalb das folgende Byte auch wirkungslos bleibt.

Durch das nächste Byte können wir den Zeichenmodus auswählen: Während der erste, JAM1 genannt, dazu führt, daß alle Linien vollständig in der festgelegten Farbe gezeichnet werden und eine Null als Eintrag erwartet, werden bei dem zweiten Modus (XOR) die Farbangaben ignoriert. Hier werden alle Punkte auf den Linien im Farbwert invertiert. Damit wird aus einem schwarzen ein weißer Punkt, aus einem roten ein blauer, etc. Diesen Modus erhalten Sie, indem Sie eine 2 in die Struktur eintragen.

Es folgt nun die Anzahl der zu verbindenden Punkte. Maximal können Sie 255 Koordinatenpaare festlegen. Der Pointer auf die Tabelle mit diesen Angaben folgt im nächsten Langwort.

Als letztes haben wir (wie üblich) einen Zeiger auf eine weitere Border-Struktur. Wenn wir diese an dem Endpunkt des aufrufenden Borders zu zeichnen beginnen lassen, können wir auf diese Weise eine beliebige Anzahl von Punkten verbinden.

Zum Schluß noch ein Wort zu der Tabelle mit den Koordinateneinträgen: Diese erwarten für jeden Punkt zwei Wörter, wobei im ersten Wort die x-Koordinate und im zweiten die y-Koordinate erwartet wird. Bezogen werden alle Angaben auf den Startpunkt, den man beim Aufruf des Borders festgelegt hat. So wird durch die unten aufgeführte Beispieltabelle ein Quadrat gezeichnet, das den Startpunkt genau in der Mitte hat.

```
koordtab    dc.w  -10,-10   ;erster Punkt (links oben)
            dc.w  10,-10    ;zweiter Punkt (rechts oben)
            dc.w  10,10     ;dritter Punkt (rechts unten)
            dc.w  -10,10    ;vierter Punkt (links unten)
            dc.w  -10,-10   ;Quadrat wird geschlossen
```

Man erkennt, daß immer eine Angabe mehr erforderlich ist, als das Polygon Eckpunkte besitzt, da es nach dem Zeichnen ja noch geschlossen werden muß.

2.10 Die Requester

Requester sind eine äußerst nützliche Angelegenheit, wenn der Anwender gezwungen werden soll, auf irgendetwas zu reagieren. Sie alle (zumindest die Leser mit nur einem Laufwerk) kennen den lästigen Requester, indem Sie aufgefordert werden »Please insert Disk xyz in drive 0«.

Ein Requester hat die Eigenschaft, immer in der linken oberen Ecke des Windows zu liegen (zumindest dort zu beginnen, Sie können es natürlich beliebig groß gestalten...). Sie können einen Hinweistext installieren und zwei Gadgets, in denen alternative Lösungsvorschläge des bestehenden Problems angeboten werden sollten (hier z.B. »Cancel« und »Retry«, es wäre gemein, zweimal »Retry« anzubieten).

Nun kann man ein Requester theoretisch auch selbst programmieren, indem man ein entsprechendes Window öffnet und in diesen zwei Gadgets installiert. Dies ist aber wenig sinnvoll, da ein Requester einem die komplette Arbeit beim Auswerten abnimmt. Es wartet so lange, bis der Anwender eines der beiden Gadgtes angeklickt hat und sagt uns dann, welches gewählt wurde. Dazu werden die Gadgets automatisch zentriert, d.h. der Abstand des linken bzw. rechten Gadgets von der linken bzw. rechten Kante ist immer identisch, egal wie groß Sie das Requester gestalten.

Da ein Requester immer nur an ein Window gebunden ist (nämlich an das, von dem es aufgerufen wurde und in dem es erscheint), werden auch nur alle weiteren Aktionen in diesem Window unterbunden, bis Sie sich für eine der beiden Auswahlmöglichkeiten entschieden haben. Sobald Sie sich aber in ein anderes Window eingeklickt haben, können Sie wieder nach Herzenslust operieren. Der Aufruf erfolgt durch die Funktion AutoRequest. Dabei müssen folgende Paramter übergeben werden:

Register	**Parameter**
A0	Zeiger auf Window-Struktur
A1	Zeiger auf Textstruktur für Text über Gadgets (»Please insert...«)
A2	Zeiger auf Textstruktur für Text linkes Gadget (»Retry«)
A3	Zeiger auf Textstruktur für Text rechtes Gadget (»Cancel«)
D0	IDCMP-Flag für linkes Gadget
D1	IDCMP-Flag für rechtes Gadget
D2	Die Breite des Requesters
D3	Die Höhe des Requesters

Mit den Registern D0 und D1 hat es folgendes auf sich: Wenn Sie in unserem Beispiel eine Diskette ins Laufwerk einschieben, ohne das Gadget »retry« anzuklicken, verschwindet das Requester wie von Geisterhand angewählt. Nun, die Sache ist überhaupt nicht geheimnisvoll. Vielmehr hat der Programmierer im Register D0 den Wert $8000 übergeben, was dem IDCMP-Flag DISKINSERTED entspricht. Dem Requester wurde dadurch mitgeteilt, daß dieses Ereignis dem Anklicken des linken Knopfes gleichkommen soll. Je mehr IDCMP-Flags Sie also übergeben, desto mehr Möglichkeiten bestehen, den Requester zu beenden. Wenn Sie eine Null übergeben (kein Flag), kann nur durch Anklicken des Gadgets beendet werden. Nun aber zu der Routine, die einen Requester aufruft:

```
move.l windowpointer,a0
move.l #bottomtext,a0
move.l #lefttext,a1
move.l #righttext,a2
move.l #$8000,d0       ;Beenden durch Anklicken und Diskette einlegen
move.l #0,d1           ;rechts nur durch Anklicken
move.l #breite,d2
move.l #höhe,d3
CALLINT AutoRequest
```

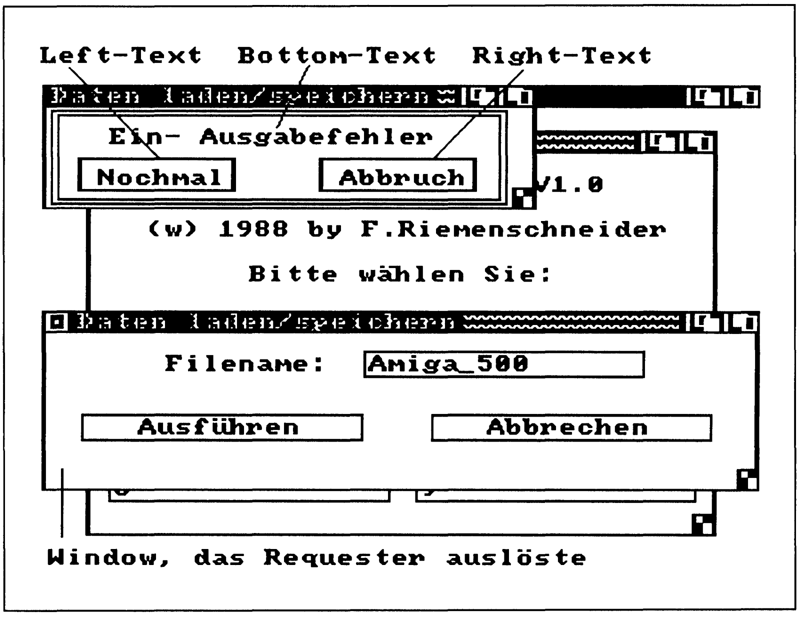


Bild 2.8: Requester nach I/O-Fehler

Damit sind wir schon am Ende der Requester und, man sehe und staune, vom Kapitel Intuition. Ich gebe zu, daß diese Thematik oft etwas trocken erscheint. Das Ergebnis, ein bedienerfreundliches, mausgesteuertes Programm entschädigt jedoch für einen noch so großen Strukturen-Urwald. Ich habe mich (erfolgreich) bemüht, alle in diesen zehn Abschnitten vorgestellten Elemente in die Beispielprogramme zu integrieren, so daß Sie sofort die Umsetzung der Theorie in die Praxis nachvollziehen können. Besonders das Zusammenspiel von vielen Elementen wird meiner Meinung nach in den meisten Büchern zu sehr vernachlässigt, wobei man bei der Gestaltung der eigenen Programme doch meistens alle benötigt, oder können Sie sich ein Programm ausschließlich mit Gadgets, ohne Menüs, Screens oder Windows vorstellen?

Rechnen mit Fließkommazahlen

Kapitel 3

Dieses Kapitel, liebe Leser, war für mich die eigentliche Motivation für die Erstellung des gesamten Buches. Als ich im März 1988 vom C64 auf den Amiga umstieg, glaubte ich, daß es ähnlich wie bei C64 reichlich Lektüre über alle relevanten Themen geben würde.

Diese Annahme stellte sich leider als sehr naiv heraus. Zwar gibt es im Juli 1988, der Zeitpunkt, an dem ich diese Sätze niederschreibe, reichlich Bücher über den Amiga und auch über die Maschinensprache, die Inhalte beschränken sich aber auf einige wenige, sich immer wiederholende Themengebiete. Die wichtigsten - und damit auch das wichtigste Thema schlechthin - wurden aber von allen Autoren sorgsam gemieden: das Rechnen mit Fließkommazahlen.

Ohne das Wissen darüber ist ein Programmierer absolut hilflos, da er 99,9 % aller vorkommenden Zahlen nicht behandeln kann. Da helfen auch keine Kurzverweise auf die Mathematik-Libraries am Ende eines Buches weiter.

Der Grund des sorgsamen Vermeidens der Fließkommazahlen ist daher wohl nicht in ihrer Bedeutung zu sehen, sondern darin, daß der Umgang mit ihnen nicht so umkompliziert ist wie mit Bytes, Wörtern und Langwörtern. Besonders die C64-Umsteiger werden sich die Haare raufen, da das Format der Amiga-Fließkommazahlen nicht mit dem des C64 übereinstimmt. Hier gilt wie fast immer, Umlernen ist schwieriger als Neulernen.

Immer wieder wird die Frage gestellt, ob es überhaupt sinnvoll ist, Rechenprogramme in Maschinensprache zu schreiben, da man dort ja auch gezwungen ist, die gleichen Routinen der Librarys zu verwenden wie es z.B. auch der Basic-Interpreter tut. Diese Frage kann man ohne Einschränkungen bejahen. Im Vergleich zum Basic kann man bei reinen Rechenroutinen von einer bis zu sechsmal schnelleren Abarbeitung ausgehen, da die Interpretationszeit der Befehle völlig wegfällt. Gegenüber C ist nur noch ein kleiner Geschwindigkeitsvorteil auszumachen, dafür kann man seine Programme (wie immer) viel kompakter schreiben.

Der Amiga bietet uns zwei verschiedene Zahlenformate an: die sogenannten FFP-(Fast-Floating-Point-)Zahlen sowie die IEEE-Zahlen mit (angeblich) doppelter Genauigkeit. Den Aufbau und die Unterschiede beider Zahlenformate werden wir im folgenden ausführlich besprechen.

3.1 Aufbau der FFP-Zahlen

Als Fließkommazahlen (die Herkunft des Namens werden wir noch erklären) bezeichnet man die Menge aller auf dem Amiga darstellbaren Zahlen. Dabei handelt es sich um einen sinnvollen Ausschnitt aus dem unendlich großen Zahlenspektrum, da man auf einem Rechner natürlich nicht beliebig große Zahlen darstellen kann. Wir werden jedoch sehen, daß dieser winzige Ausschnitt normalerweise ausreicht, es sei denn, man rechnet mit Zahlen, wie sie Dagobert Duck zum Zählen seiner Geldmengen benötigt. Fließkommazahlen kann man nicht wie z.B. Integerzahlen in ein bestimmtes Schema pressen. Eine Fließkommazahl kann auch eine Integerzahl sein, ebensogut jedoch auch eine positive oder negative gebrochene Zahl oder eine ganze Zahl, die außerhalb des Integer-Bereiches liegt. Eigentlich ist auch die Menge der Fließkommazahlen unendlich groß, da man für zwei beliebig dicht zusammenliegende Zahlen immer noch eine finden kann, die zwischen diesen beiden liegt. Da jedoch auch die Rechengenauigkeit eines Computers begrenzt ist (auch darauf kommen wir noch genauer zu sprechen), ist die Zahl der darstellbaren Zahlen doch eingeschränkt, wenn auch die Anzahl der möglichen Werte sehr groß ist. Daraus kann man schon erahnen, daß die Darstellung im Rechner nicht so einfach ist wie zum Beispiel die der Integerzahlen.

Der Mensch rechnet im Gegensatz zum Computer normalerweise im Dezimalsystem, welches das Zahlensystem mit der Basis 10 ist. Wenn man eine bestimmte Zahl darstellen möchte, teilt man diese (wenn auch unbewußt) in die sogenannte Mantisse und den Exponent zur Basis 10 auf. Die Mantisse ist dabei der Vorfaktor, der angibt, wie oft man die jeweilige Zehnerpotenz hernehmen muß, um die gewünschte Zahl zu erhalten:

$7 = 7 * 10^0$
$31 = 3,1 * 10^1$
$1.111.000 = 1.111 * 10^6$

Man sieht jedoch, daß diese Zahlendarstellung nicht eindeutig ist; man kann ein und dieselbe Zahl auf beliebig viele Arten mit jeweils unterschiedlichem Exponenten darstellen:

$31 = 0,31 * 10^2 = 3,1 * 10^1 = 31 * 10^0$ usw.

Deshalb hat man sich auf die sogenannte normalisierte Darstellung geeinigt. Dabei kann die Mantisse nur einen Wert zwischen 1 und der Basis des Zahlensystems, hier also 10, annehmen. In unserem Beispiel wäre also die $3,1 * 10^1$ die normalisierte Darstellung. An dieser Stelle erkennt man, warum die Zahlen »Fließkommazahlen« genannt werden: Durch Verschiebung des Kommas (dies kann durch alle Stellen der Mantisse »fließen«) und gleichzeitige Änderung des Exponenten kann man eine Zahl auf verschiedene Weisen darstellen. Das Komma hat also keinen festen Platz, sondern ändert diesen in Abhängigkeit der Zehnerpotenz. Bei einer Erhöhung des Exponenten um eins muß es um eine Stelle nach links verschoben werden, bei einer Verschiebung um eine Stelle nach rechts muß der Exponent um eins erniedrigt werden.

Mit ausschließlich positivem Exponenten kann man in der normalisierten Form nur Zahlen, die größer oder gleich eins sind, darstellen, da der kleinste Wert $1 * 10^0$ ($10^0 = 1$!) ist. Um kleinere Zahlen verwenden zu können, muß man auch negative Exponenten zulassen:

$0{,}9 = 9 * 10^{-1}$

Negative Zahlen werden einfach durch ein Minussetzen der Mantisse erreicht, so daß man auf diese Weise tatsächlich das gesamte Zahlenspektrum abdecken kann. Durch Anwendung der Potenzgesetze kann man zu rechnen beginnen. Um zwei Dezimalzahlen beispielsweise zu multiplizieren, muß man die Mantissen multiplizieren und die Exponenten addieren:

$3{,}5 * 12 = (3{,}5 * 10^0) * (1{,}2 * 10^1) = 4{,}2 * 10^1 = 42$

Bei der Addition kann man nur Zahlen mit gleichem Exponenten verwenden, so daß möglicherweise normalisierte Zahlen in eine andere Darstellung durch Verschiebung des Kommas der Mantisse umgewandelt werden müssen:

$$
\begin{aligned}
2{,}5 + 11 &= (2{,}5 * 10^0) + (1{,}1 * 10^1) \\
&= (0{,}25 * 10^1) + (1{,}1 * 10^1) \\
&= 1{,}35 * 10^1 = 13{,}5
\end{aligned}
$$

Der entscheidende Vorteil dieser Darstellungsweise liegt darin, daß man jede Dezimalzahl in einem überschaubaren Format darstellen kann. Für den Computer ist es ja unerläßlich, daß jede Zahl, egal wie viele Stellen diese normalerweise besitzt oder wie groß sie ist, in einem bestimmten Format gespeichert werden kann, da eine Verarbeitung sonst unmöglich ist. Wir sehen also, daß wir im Dezimalsystem durch Verwendung von Mantisse und Exponent gute Ergebnisse erzielen können. Leider rechnet unser Amiga jedoch im Binärsystem, also dem Zahlensystem mit der Basis zwei. Wir müssen nun sehen, wie wir diese Darstellungsweise auf das Binärsystem übertragen können. Nun, versuchen wir doch einmal, eine Binärzahl in Mantisse und Exponenten aufzuteilen:

$1.1011 * 10^{1010} = 1.1011 * 10^{10}$ (binär)

$$
\begin{aligned}
&= 1 * 10^{10} + 1 * 10^9 + 0 * 10^8 + 1 * 10^7 + 1 * 10^6 \\
&= \quad 1024 + \quad 512 + \quad 0 \quad + \quad 128 \quad + \quad 64 \\
&= 1728
\end{aligned}
$$

Auch gebrochene Binärzahlen sind unproblematisch:

$1.101 * 10^0$

$$
\begin{aligned}
&= 1 * 10^0 + 1 * 10^{-1} + 0 * 10^{-2} + 1 * 10^{-3} \\
&= 1 \quad + \quad 0.5 \quad + \quad 0 \quad + \quad 0.125 \\
&= 1.625
\end{aligned}
$$

In der Praxis brauchen wir aber den umgekehrten Weg, d.h., wir müssen die uns vertrauten Dezimalzahlen in Binärzahlen umwandeln. Dazu gibt es ein Verfahren, das ich an einem Beispiel (1965,125) verdeutlichen möchte. Zunächst wird die Zahl in den Vorkomma- (1965) und in den Nachkommaanteil (0,125) aufgeteilt. Der Vorkommaanteil wird mit Restbildung so lange durch zwei geteilt, bis die Null erreicht wird:

1965	: 2 =	982	Rest 1
982	: 2 =	491	Rest 0
491	: 2 =	245	Rest 1
245	: 2 =	122	Rest 1
122	: 2 =	61	Rest 0
61	: 2 =	30	Rest 1
30	: 2 =	15	Rest 0
15	: 2 =	7	Rest 1
7	: 2 =	3	Rest 1
3	: 2 =	1	Rest 1
1	: 2 =	0	Rest 1

Man erhält nun die Dezimalzahl einfach dadurch, daß man die Reste von unten nach oben als Binärziffern interpretiert:

$$\begin{aligned} 11110101101 &= 2^{10} + 2^9 + 2^8 + 2^7 + 2^5 + 2^3 + 2^2 + 2^0 \\ &= 1024 + 512 + 256 + 128 + 32 + 8 + 4 + 1 \\ &= 1965 \end{aligned}$$

Der Nachkommaanteil wird genau umgedreht behandelt: Statt durch zwei zu dividieren, wird er so lange mit zwei multipliziert, bis der Nachkommaanteil des Ergebnisses Null wird. Die Vorkommastellen des Ergebnisses werden jedesmal abgeschnitten und notiert:

0,125	* 2 = 0,25	Vorkommastelle 0
0,25	* 2 = 0,5	Vorkommastelle 0
0,5	* 2 = 1,0	Vorkommastelle 1

Die gesuchte Binärzahl ergibt sich aus den Vorkommastellen von oben nach unten gelesen:

$$0{,}001 = 0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 0{,}125$$

Die Zahl 1965,125 wird durch die Binärzahl 11110101101,001 repräsentiert. Da die Zahl 1965,125 auch als 1965,125 * 10^0 geschrieben werden kann, vor der Umwandlung also nicht normalisiert wurde, ist das Ergebnis auf den Exponenten Null bezogen:

$$11110101101{,}001 = 11110101101{,}001 * 2^0$$

Es ist aber auch im Binärsystem möglich, eine Normalisierung durchzuführen. Wie oben erwähnt, muß sich die Mantisse dabei im Bereich zwischen 1 und der Basis des Zahlensystems, also 2, bewegen. Da diese selbst jedoch ausgeschlossen bleibt, kann eine normalisierte Mantisse im Dualsystem nur eine Eins vor dem Komma haben, andere Werte sind nicht möglich. Dies ist für die IEEE-Darstellung, wie wir noch sehen werden, von Bedeutung. In unserem Beispiel wäre die normalisierte Form der Zahl

$$11110101101001 * 2^{10}$$

Interessant ist jedoch, daß es nicht immer möglich ist, eine Dezimalzahl in eine Dualzahl umzuwandeln, z.B. 0,3. Nach obigem Verfahren geht es los:

0,3 * 2 = 0,6 Vorkommastelle 0
0,6 * 2 = 1,2 Vorkommastelle 1
0,2 * 2 = 0,4 Vorkommastelle 0
0,4 * 2 = 0,8 Vorkommastelle 0
0,8 * 2 = 1,6 Vorkommastelle 1

An dieser Stelle wiederholt sich der Vorgang ab der zweiten Zeile, es wird nie ein Nachkommaanteil Null erreicht. Die Dualzahl würde daher durch eine Periode

$$0,\overline{1001}$$

angenähert werden können. Dies ist jedoch kein Phänomen der Umwandlung ins Dualsystem. Die Zahl $^1/_3$ kann z.B. im Dreiersystem ohne weiteres als $1 * 3^{-1}$ dargestellt werden, während die Umwandlung ins Dualsystem versagt:

$^1/_3 = 0,333$

Bevor wir nun auf den Amiga zu sprechen kommen, müssen wir uns noch über Zahlen, die kleiner als eins sind, Gedanken machen, da hier offenbar ein negativer Exponent erforderlich ist. Da im Zweiersystem keine negativen Zahlen existieren, müssen wir ein anderes Verfahren anwenden. Wenn wir davon ausgehen, daß wir für den Exponenten 7 Bit reservieren, könnten wir das oberste sechste Bit als Vorzeichenbit interpretieren, wie dies auch bei den 8-Bit-Integerzahlen geschieht. Dabei wird ein gesetztes Bit 7 als Zeichen für eine negative Zahl verwendet (siehe Kapitel 1). Damit kann man Exponenten von -64 bis +63 darstellen. Hier erkennt man nun auch, daß der Exponent für den eingangs erwähnten Bereich der benutzbaren Zahlen verantwortlich ist. Bei Verwendung von 7 Bit für den Exponenten kann man somit Zahlen

von $2^{-64} = 5,421 * 10^{-20}$ bis $2^{63} = 9,22 * 10^{18}$

darstellen, was in fast allen Anwendungsfällen genügen dürfte. Die C64-Umsteiger unter den Lesern werden jetzt aber den Kopf schütteln, zumindest die, die das Buch »C64/C128: Alles über Maschinensprache« gelesen haben. Der C64 stellt nämlich für den Exponenten 1 Bit mehr zur Verfügung, so daß sich der Rechenbereich

von $2^{-128} = 2,938 * 10^{38}$ bis $2^{127} = 1,701 * 10^{38}$

erstreckt. Die Antwort lautet, daß das siebte Bit beim Amiga FFP-Format noch für andere Zwecke benötigt wird; dazu komme ich aber später. Der Amiga arbeitet jedoch genau wie der C64 ohne Vorzeichenbit, statt dessen addiert er zu jedem Exponenten einen Offset von 65 (129 beim C64) und interpretiert die so entstandene Zahl als positiv und vorzeichenlos. Damit werden die tatsächlichen Exponenten -64 bis +62 in die Zahlen 1 bis 127 umgewandelt. Der Exponent +63 kann in diesem Format nicht mehr berücksichtigt werden, während die Null unbenutzt bleibt. Der Grund hierfür liegt darin, daß bisher eine wichtige Zahl überhaupt noch nicht behandelt wurde: die Null! In der Exponentenschreibweise

kommt man nicht an sie heran, so daß ihr vereinbarungsgemäß der Exponent Null zugeordnet wird:

Exponent	Darstellung	Wert
0	0	0
-64	1	$5.42 * 10^{-20}$
-63	2	$1.08 * 10^{-19}$
-62	3	$2.17 * 10^{-19}$
-61	4	$4.33 * 10^{-19}$
-2	63	0.25
-1	64	0.5
+-0	65	1
+1	66	2
+2	67	4
+58	123	$2.88 * 10^{17}$
+59	124	$5.76 * 10^{17}$
+60	125	$1.15 * 10^{18}$
+61	126	$2.31 * 10^{18}$
+62	127	$4.61 * 10^{18}$

Nachdem wir also festgestellt haben, daß der Exponent für den Rechenbereich verantwortlich ist, kommen wir zu der Mantisse, die über die Rechengenauigkeit entscheidet. Der Amiga verwendet für sie im FFP-Format 24 Bit. Dieses Ergebnis ist für alle Leser, die jemals einen C64 besessen haben, schockierend, wenn sie wissen, daß der C64 32 Bit verwendet hat und dabei sehr ungenaue Rechnungen ausführte.

Der Grund, beim Amiga nur 24 Bit Mantisse zuzulassen, besteht darin, daß man so eine komplette FFP-Zahl in ein Prozessorregister (32 Bit) einlesen kann. Während es bei der Bytemaschine C64 nicht weiter darauf ankam, ob man nun 4 oder 5 Byte für eine Fließkommazahl benutzte, würde diese Darstellung auf dem Amiga dazu führen, daß man eine Fließkommazahl nicht mehr mit nur einem Maschinenbefehl bewegen kann. Deshalb ist die Kürzung der Mantisse um 8 Bit durchaus gerechtfertigt. Sie wird linksbündig dargestellt, d.h., alle nicht benötigten Bits werden mit Nullen aufgefüllt. Mantissen, die mehr als 24 Bit aufweisen, werden natürlich abgeschnitten.

Da das oberste Bit ja die Vorkommastelle der Mantisse darstellt, die immer eins ist (siehe oben), ist das unterste Mantissenbit für den Wert 2^{23} verantwortlich, z.B. Mantissenbit:

```
23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 1  0  0  1  1  1  0  1  0  0  1  0  1  0  1  0  1  0  1  1  1  0  0  1
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '  '

 0  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9  1  1  1  1  1  1  1  1  1  1  2  2  2  2
                               0  1  2  3  4  5  6  7  8  9  0  1  2  3
```

Damit kann man Zahlen, die sich um den Wert 2^{-23} unterscheiden, gerade noch differenzieren. Dabei handelt es sich um den Wert

$1{,}192 * 10^{-7} = 0{,}0000001192$

Der Amiga besitzt also eine Rechengenauigkeit von sechs Stellen, von der siebten Stelle an wird ungenau gerechnet. Das Problem bei den Rundungsfehlern liegt darin, daß der Fehler mit der Anzahl der Rechenoperationen immer größer wird. Allein durch die natürliche Rechengenauigkeit ließen sich aber beim C64 (9-Stellen-Genauigkeit) die oft eklatanten Fehler nicht erklären. Die Hauptschuld trugen nämlich die unzureichend programmierten Rechenroutinen, die viel ungenauer arbeiteten als neun Stellen. So wurden Sinus und Cosinus durch Polynome angenähert, die von der 9. Stelle hinter dem Komma nur träumen konnten. Für uns Amiga-Anwender bleibt die Hoffnung, daß die Math-Librarys bessere Rechenroutinen enthalten, die zumindest die natürliche Genauigkeit von sechs Stellen erreichen. Genauer gesagt muß es heißen: Uns blieb die Hoffnung. In der Tat sind die FFP-Rechenroutinen des Amiga ebenso ungenau und teilweise noch ungenauer als die des C64. In unserem ersten Beispielprogramm werden wir dies noch zu spüren bekommen. Es ist schon etwas erschütternd, wenn ein so großartiger Rechner wie der Amiga sich in der Rechengenauigkeit von einem Gerät wie dem C64 schlagen läßt.

Nun, bis auf ein Problem haben wir alles gelöst: Dieses Problem stellen die negativen Zahlen dar. Wie kann man ein negatives Mantissenvorzeichen darstellen? Man benutzt einfach wie beim Exponenten ein weiteres Vorzeichenbit. Nun könnte man einfach wie beim C64 auf die geniale Idee kommen, das oberste Mantissenbit durch dieses Vorzeichen zu ersetzen. Dies ist zulässig, da jenes Bit ja immer 1 ist und deshalb keinen Informationsgehalt besitzt. Leider stellt das FFP-Format des Amiga auch hier einen Rückschritt dar: Statt des eben beschriebenen, eleganten Verfahrens wird ein wirklich plumpes verwendet: Da in dem Exponentenbyte ja noch ein Bit frei ist (das siebte und oberste), kann man hier bequem das Mantissenvorzeichenbit einblenden. Sie sehen, liebe Leser, mit etwas gutem Willen hätten wir auch einen 8-Bit-Exponenten und damit einen wesentlich erweiterten Rechenbereich haben können. So gesellt sich aber zu der Frustration über die Rechengenauigkeit eine weitere hinzu. Der prinzipielle Aufbau einer FFP-Zahl sieht somit wie folgt aus:

Bits

```
3322222222221111111111
10987654321098765432109 8 7 6543210
|                        |N|  Exp  |
|    Mantisse 0-23       |E|       |
|                        |G|  1-7  |
```

Zum Schluß möchte ich Ihnen das bislang Gesagte durch drei Beispiele verdeutlichen.

Dezimalzahl	Fließkommazahl (FFP-Format)	
	Binär	Hexadezimal
0 10 -10	00000000000000000000000000000000 10100000000000000000000001000100 10100000000000000000000011000100	\$00000000 \$A0000044 \$A00000C4

3.2 Aufbau der IEEE-Zahlen

Die zweite Darstellungsart der Fließkommazahlen wird auch die mit »doppelter Genauigkeit« genannt. Dieser Begriff ist jedoch sachlich falsch, wie wir gleich sehen werden. Gegenüber den FFP-Zahlen hat sich die prinzipielle Art, eine Fließkommazahl darzustellen, nicht geändert. Modifiziert wurden nur die Dimensionen von Mantisse und Exponent sowie der gestiegene Platzbedarf um ein weiteres Langwort auf 64 Bit. Der Exponent beinhaltet davon 11 Bit und belegt die Bits 30–20 des ersten Langwortes. Damit können die vorzeichenbehafteten Zahlen -1024 bis +1023 verwendet werden. Der Wertebereich erstreckt sich somit

von $2^{-1024} = 8{,}636 * 10^{-312}$ bis $5{,}790 * 10^{307}$

Sie sehen ein, liebe Leser, daß man mit diesem gigantischen Wertebereich wirklich jedes Problem lösen kann. Der Exponent wird wiederum aber nicht als vorzeichenbehaftete, sondern durch Addition eines Offsets als positive, vorzeichenlose Zahl dargestellt, damit man auch die Zahl Null behandeln kann. Dafür wird bei den IEEE-Zahlen der 12-Bit-Wert $3FF (1023) hergenommen. Damit verschiebt sich die Zahlenmenge des Exponenten nach -1022 bis +1024. Somit ergibt sich zwischen tatsächlichem Exponenten und seiner Darstellung folgender Zusammenhang:

Exponent	Darstellung	Wert
0	0	0
-1022	1	$2.16 * 10^{-312}$
-1021	2	$4.32 * 10^{312}$
-1020	3	$8.64 * 10^{-312}$
-1019	4	$1.73 * 10^{-311}$
-2	1021	0.25
-1	1022	0.5
+-0	1023	1
+2	1025	4
+1020	2043	$1.16 * 10^{308}$
+1021	2044	$2.32 * 10^{308}$
+1022	2045	$4.63 * 10^{308}$
+1023	2046	$9.26 * 10^{308}$
+1024	2047	$1.85 * 10^{309}$

Das 31. Bit des ersten Langwortes wird als Vorzeichenbit der Mantisse verwendet. Die Mantisse besteht aus 53 Bit. Jetzt denken Sie sicherlich, ich habe mich verschrieben, da 12 von 64 Bit verbraucht wurden und somit nur noch 52 übrigbleiben. Nein, es sind 53 Bit, da nun endlich das vom C64 bekannte Verfahren übernommen wurde, das Vorkommabit der Mantisse nicht mehr darzustellen, da es immer eins und somit überflüssig ist.

Etwas ungewöhnlich ist die Reihenfolge der einzelnen Mantissenbits: Nachvollziehbar ist noch, daß zunächst die Bits 19–0 des ersten Langwortes verwendet werden. Dann jedoch geht es im zweiten Langwort nicht etwa mit dem 31., sondern mit dem 15. Bit weiter. Die folgenden 16 Mantissenbits werden so durch die Bits 15–0, erst die letzten 16 durch die Bits 31-16 des zweiten Langwortes repräsentiert :

```
                    3 3222222222 21111111111
Bits 1. Langwort:   1 0987654321 09876543210987654321 0
                   |N|          |                    |
(Mantisse 0 =1)    |E| Exponent |  Mantisse 1-20     |
                   |G|          |                    |

                    3322222222221111 111111
Bits 2. Langwort:8765432109876 5432109876543210
                   |                |                |
                   | Mantisse 37-52 | Mantisse 21-36 |
```

Wie oben schon erwähnt, stellt das erste Mantissenbit bereits das erste Nachkommabit dar, was am Beispiel der Zahl 3 erläutert werden soll:

$3 = 2^1 + 2^0 = 1{,}1 * 2^1$

Erwartete Darstellung:

```
N| Exponent  | Mantisse .......
0 10000000000 110000000 .......
```

Tatsächliche Darstellung:

```
N| Exponent  | Mantisse .......
0  1000000000 100000000 .......
```

Das gesetzte erste Mantissenbit entspricht dem 2^0, das Vorkommabit (2^1) wird automatisch als gesetzt angenommen.

Nun zur Rechengenauigkeit der IEEE-Zahlen: Mit real 53 Mantissenbits kann man Zahlen, die sich um den Wert 2^{-52} $^{-52}$ differieren, gerade noch so eben unterscheiden. Hierbei handelt es sich um den Wert

$2^{-52} = 2{,}22 * 10^{-16} = 0{,}00000000000000022$

Damit wird eine Genauigkeit von 15 Stellen erreicht, von der 16. Stelle an wird ungenau gerechnet. Gegenüber der Genauigkeit der FFP-Zahlen von sechs Stellen konnte diese also um den Faktor 2,5 gesteigert werden, weshalb die Bezeichnung »Zahlen mit doppelter Genauigkeit« auch nicht zutrifft, da sie dem wirklichen Leistungsumfang der IEEE-Zahlen nicht gerecht wird. Gegen die Verwendung der IEEE-Zahlen sprechen jedoch zwei wesentliche Argumente: Zum einen belegen sie doppelt soviel Speicherplatz wie die FFP-Zahlen, zum anderen laufen die Rechenoperationen gegenüber diesem Format (noch) langsamer ab. Wir werden uns in unseren Beispielprogrammen deshalb auf die FFP-Zahlen beschränken, zumal die Quelltexte auch so schon lang genug sind.

3.3 Rechnen mit der MATHFFP-Library

Diese Library stellt einfache Rechenoperationen wie Addition, Subtraktion, Multiplikation und Division zur Verfügung. Daneben ist es noch möglich, das Vorzeichen einer FFP-Zahl zu wechseln und den Betrag zu bilden. Abgerundet wird die Library durch die Umwandlungsroutinen vom Langwort ins FFP-Format und umgedreht. Bevor die Routinen benutzt werden können, muß das Include-File math/mathffp_lib.i geladen werden, wie es in Kapitel 1 beschrieben wurde.

a) Langwort in FFP-Format wandeln: **SPFlt**
Diese Funktion dient dazu, eine vorzeichenbehaftete 32-Bit-Zahl (siehe Kapitel 1) in eine Fließkommazahl des FFP-Formates umzuwandeln. Wir wollen dies mit der Zahl -2 tun:

```
move.l #-2,d0
CALLFFP SPFlt
```

Ergebnis in D0 : `$800000C2`

b) FFP-Zahl in Langwort wandeln: **SPFix**
Hiermit können Sie eine Fließkommazahl in ein vorzeichenbehaftetes Langwort umwandeln. Dabei werden eventuelle Nachkommastellen abgeschnitten. Ein Beispiel für die Zahl 12,5:

```
move.l #$C8000044,d0
CALLFFP SPFix
```

Ergebnis in D0 : `$0000000C`

c) Vergleich zweier FFP-Zahlen : **SPCmp**
Fließkommazahlen kann man nicht einfach mit den Vergleichsbefehlen des Prozessors vergleichen, da sowohl Exponent und Mantisse verglichen werden müssen. Die SPCmp-Routine liefert abhängig vom Ergebnis drei verschiedene Ergebnisse zurück, wobei die zu vergleichenden Zahlen in den Registern D0 und D1 stehen müssen:

```
:  0, falls FFP(D0) gleich FFP(D1)
: +1, falls FFP(D0) kleiner FFP(D1)
: -1, falls FFP(D0) größer FFP(D1)
```

(Die Aussagen im Amiga-Programmierhandbuch auf Seite 316 sind falsch!)

Der Vergleich läuft im Programm so ab:

```
move.l zahl1,d0
move.l zahl2,d1
CALLFFP SPCmp
tst.l d0              ;Flags setzen
beq gleich            ;Zero-Flag, dann ist D0 gleich Null
bmi größer            ;Negativ-Flag, dann ist D0 negativ
kleiner .....         ;gar nichts, dann ist D0 positiv

gleich  .....

größer .....
```

d) Testen einer FFP-Zahl: **SPTst**
In Punkt c) haben wir den Prozessorbefehl TST dazu verwendet, eine Integerzahl in die Rubriken gleich Null, kleiner als Null oder größer als Null einzustufen. Genau dieselbe Wirkung wird durch die SPTst-Funktion bezüglich den Fließkommazahlen bewirkt. Als Ergebnis kommt zurück:

```
:  0, falls Zahl gleich Null ist
: -1, falls Zahl kleiner als Null ist
: +1, falls Zahl größer als Null ist
```

Die Auswertung erfolgt analog zu Punkt c):

```
move.l #zahl,d0
CALLFFP SPTst
tst.l d0            ;Flags setzen
beq gleich          ;Zero-Flag, dann ist D0 gleich Null
bpl größer          ;kein Negativ-Flag, dann ist D0 positiv
kleiner .....       ;gar nichts, dann ist D0 negativ

gleich  .....
grösser .....
```

e) Absolutwert einer FFP-Zahl: **SPAbs**
Man kann auch Betrag dazu sagen. Wir wollen den Absolutwert der Zahl -3 bestimmen:

```
move.l #$C00000C2,d0        ;-3 im FFP-Format
CALLFFP SPAbs
```

Ergebnis in D0: `$C0000042   ;+3 im FFP-Format`

f) Negatbildung einer FFP-Zahl: **SPNeg**
Die Bildung des Negats bedeutet, daß eine positive Zahl in eine negative umgewandelt wird und umgedreht. Auf die Zahl Null hat diese Funktion keinen Einfluß. Die Bildung des Negats von 7 läuft wie folgt ab:

```
move.l #$E0000043,d0        ;7 im FFP-Format
CALLFFP SPNeg
```

Ergebnis in D0: `$E00000C3    ;-7 im FFP-Format`

g) Addition zweier FFP-Zahlen: **SPAdd**
Die beiden FFP-Zahlen in den Registern D0 und D1 werden addiert. Als Beispiel sollen die Zahlen -4 und +6 addiert werden:

```
move.l #$800000C3,d0        ;-4 im FFP-Format
move.l #$C0000043,d1        ;+6 im FFP-Format
CALLFFP SPAdd
```

Ergebnis in D0: `$80000042    ;+2 im FFP-Format`

h) Subtraktion zweier FFP-Zahlen: **SPSub**
Die FFP-Zahl im Register D1 wird von der in Register D0 abgezogen. Wir wollen die Zahl 3 von der Zahl -1 abziehen:

```
move.l #$800000C1,d0        ;-1 im FFP-Format
move.l #$C0000042,d1        ;3 im FFP-Format
CALLFFP SPSub
```

Ergebnis in D0: `$800000C3    ;-4 im FFP-Format`

i) Multiplikation zweier FFP-Zahlen: **SPMul**
Die beiden Fließkommazahlen in den Registern D0 und D1 werden miteinander multipliziert, z.B. 4 mal 0,25 :

```
move.l #$80000043,d0        ;4 im FFP-Format
move.l #$8000003F,d1        ;0.25 im FFP-Format
CALLFFP SPMul
```

Ergebnis in D0: `$80000041    ;1 im FFP-Format`

j) Division zweier FFP-Zahlen: **SPDiv**
Die Fließkommazahl in Register D0 wird durch diejenige in D1 dividiert. Als Beispiel soll 2 durch -4 dividiert werden:

```
move.l #$80000042,d0        ;2 im FFP-Format
move.l #$800000C3,d1        ;-4 im FFP-Format
CALLFFP SPDiv
```

Ergebnis in D0: `$800000C0    ;-0.5 im FFP-Format`

Die folgenden beiden Funktionen mit den rätselhaften Namen Ceil (Decke) und Floor (Boden) wurden neu in die Kickstart-Version 1.2 aufgenommen. Mit ihnen kann man Rundungsoperationen von FFP-Zahlen durchführen, wobei wahlweise auf- (»Decke«) oder abgerundet (»Boden«) werden kann.

k) Aufrundung auf eine ganze Zahl : **SPCeil**
Durch diese Funktion wird erreicht, daß eine FFP-Zahl auf die nächsthöhere ganze Zahl gerundet wird. Bei negativen Zahlen ist dies die betragsmäßig kleinere, folgende Zahl, z.B. -6 bei einer Rundung von -6,5. Wir wollen nun einmal die Zahl +6,5 behandeln:

```
move.l #$D0000043,d0         ;6.5 im FFP-Format
CALLFFP SPCeil
```

Ergebnis in D0: `$E0000043    ;7 im FFP-Format`

l) Abrundung auf eine ganze Zahl: **SPFloor**
Diese Funktion entspricht der INT-Funktion des C64, d.h., bei positiven Zahlen werden die Nachkommastellen abgeschnitten, bei negativen wird der betragsmäßig höhere Wert ausgegeben, z.B. -7 bei einer Rundung von -6.5. Auch hier das Beispiel für die Zahl +6,5:

```
move.l #$D0000043,d0          ;6,5 im FFP-Format
CALLFFP SPFloor
```
Ergebnis in D0: `$C0000043          ;6 im FFP-Format`

3.4 Rechnen mit der MATHTRANS-Library

Diese Library benutzt ebenfalls das FFP-Format und stellt vornehmlich transzendente Funktionen wie Sinus und Cosinus zur Verfügung. Daneben werden aber auch anspruchsvolle arithmetische Funktionen wie Potenzierung sowie Umwandlungsfunktionen in das IEEE-Format angeboten.

Die transzendenten Funktionen benötigen das Funktionsargument im Bogenmaß. Für diese Umrechnung gilt:

2*PI = 360 Grad

Sehr oft liegt jedoch ein Winkel im Gradmaß vor, so daß er vor Beginn der Rechenoperation ins Bogenmaß umgewandelt werden muß. Anschließend kann man das Ergebnis wieder ins Gradmaß zurückrechnen. Für die Umrechnung vom Grad- ins Bogenmaß gilt folgende Formel:

Winkel(Bogenmaß) = (Winkel(Gradmaß)/360)*2*PI

Im Gegensatz zum C64, wo Konstanten wie PI bereits im Fließkommaformat im ROM gespeichert sind, müssen wir uns beim Amiga selbst helfen. Da man neben der Konstanten 2*PI häufig auch 3*PI/2, PI und PI/2 benötigt, möchte ich Ihnen an dieser Stelle die entsprechenden Näherungen im FFP-Format vorstellen:

```
PId2:    $C90FDA41    ;PI/2
PI:      $C90FDA42    ;PI
PIm3d2:  $96CBE543    ;PI*3/2
PIm2:    $C90FDA43    ;PI*2
```

Der außerdem noch benötigte Wert 360 wird im FFP-Format als

```
dreihs :  $B4000049    ;360
```

dargestellt. Damit können wir die Umwandlung vollziehen, vorausgesetzt, der Winkel im Gradmaß liegt bereits im Fließkommaformat vor. Ist dies nicht der Fall, müssen Sie, bevor Sie die folgende Routine benutzen, noch die Funktion SPFlt des letzten Abschnittes vorausschicken.

```
move.l #grad,d0         ;Winkel (Grad) im FFP-Format
move.l dreihs,d1        ;360 im FFP-Format
CALLFFP SPDiv           ;Dividieren, Ergebnis in D0
move.l PIm2,d1          ;2*PI im FFP-Format
CALLFFP SPMul           ;Multiplizieren
```

Ergebnis in D0 : Winkel im Bogenmaß (FFP-Format)

Die Umrechnung vom Bogenmaß ins Gradmaß sieht so aus:

Winkel(Gradmaß) = (Winkel(Bogenmaß)/(2*PI))*360

Hier wiederum eine Umrechnungsroutine in Assembler:

```
move.l #bogen,d0        ;Winkel (Bogenmaß) im FFP-Format
move.l dreihs,d1        ;360 im FFP-Format
CALLFFP SPMul           ;Multiplizieren, Ergebnis in D0
move.l PIm2,d1          ;2*PI im FFP-Format
CALLFFP SPDiv           ;Dividieren
```

Ergebnis in D0: Winkel im Gradmaß (FFP-Format)

Nun können wir die transzendenten Funktionen berechnen, vorausgesetzt, Sie haben das Include-File math/mathtrans_lib.i geladen, wie es in Kapitel 1 beschrieben wurde.

a) Sinus einer FFP-Zahl: **SPSin**
Der Sinus des Argumentes im Register D0 wird gebildet. Da der Wert durch ein Polynom angenähert wird, kann nicht mit großartiger Genauigkeit gerechnet werden.

```
move.l #winkel,d0     ;Winkel im Bogenmaß
CALLMATHTRANS SPSin
```

Ergebnis in D0: Sinus von "winkel" im FFP-Format
Da die Aufrufsyntax der nun folgenden zehn transzendenten Funktionen mit der des Sinus identisch ist, möchte ich Ihnen nur den Makroaufruf ohne Beispiel darlegen:

b) CALLMATHTRANS SPCos : Cosinus-Funktion
c) CALLMATHTRANS SPAsin : ArcusSinus-Funktion
d) CALLMATHTRANS SPAcos : ArcusCosinus-Funktion
e) CALLMATHTRANS SPTan : Tangens-Funktion
f) CALLMATHTRANS SPAtan : ArcusTangens-Funktion
g) CALLMATHTRANS SPSinh : SinusHyperbolikus-Funktion
h) CALLMATHTRANS SPCosh : CosinusHyperbolikus-Funktion
i) CALLMATHTRANS SPTanh : TangensHyperbolikus-Funktion
j) CALLMATHTRANS SPExp : Exponentialfunktion zur Basis e
k) CALLMATHTRANS SPLog : Natürlicher Logarithmus (Basis e)
l) CALLMATHTRANS SPLog10 : Logarithmus zur Basis 10
m) Potenzierung zweier FFP-Zahlen : SPPow

Einen meiner traurigsten Momente mußte ich erleben, als ich diese Funktion zum erstenmal einsetzte: Obwohl ich bereits im Jahre 1985 die Firma Commodore auf einen Fehler in der Potenzierungsroutine des C64 aufmerksam machte, wurde dieser nicht korrigiert, sondern sogar auf den Amiga übernommen! Dies kann man wirklich nicht mehr hinnehmen. Der Fehler äußert sich dahingehend, daß negative Zahlen nicht korrekt mit einem ungeraden Exponenten potenziert werden können.

Wie jeder weiß, wird das Ergebnis einer Potenzierung einer negativen Zahl mit einem geraden Exponenten immer positiv (z.B. $-2^2 = +4$), während bei einer Potenzierung mit einem ungeraden Exponenten das negative Vorzeichen erhalten bleibt (z.B.$-2^3 = -8$!). Die Commodore-Potenzierungsroutinen liefern aber in jedem Fall ein positives Ergebnis zurück ($-2^2 = +8$!!).

Wie kann man nun diesen Fehler ausgleichen? Nun, zunächst stellt man fest, ob die Basis negativ ist. Wenn dies nicht zutrifft, kann man beruhigt weiterrechnen. Wenn sie jedoch negativ ist, muß man prüfen, ob der Exponent eine gerade Zahl ist. Dieses kann man z.B. so ausführen, daß man ihn durch 2 teilt und anschließend in eine Integerzahl umwandelt, so daß ein etwaiger Nachkommaanteil abgeschnitten wird. Dann wandelt man diese Zahl wieder ins FFP-Format, multipliziert sie mit 2 und vergleicht sie mit der ursprünglichen Fließkommazahl: Stimmen beide überein, ist der Exponent gerade und man braucht nichts zu unternehmen. Wenn er sich als ungerade herausstellt, muß man in das Ergebnis der Potenzierung anschließend ein Negativ-Bit einblenden. Zunächst möchte ich Ihnen den Beweis erbringen, daß das Prüfverfahren für gerade/ungerade wirklich funktioniert:

gerade Zahl z.B. 6

Schritt 1: Division durch 2	(Ergebnis 3)
Schritt 2: Umwandlung in Integer-Zahl	(Ergebnis 3)
Schritt 3: Umwandlung in FFP-Zahl	(Ergebnis 3)
Schritt 4: Multiplikation mit 2	(Ergebnis 6)
Schritt 5: Vergleich mit Ursprungszahl	

Beide sind identisch, daher ist 6 eine gerade Zahl.

ungerade Zahl z.B. 11

Schritt 1: Division durch 2	(Ergebnis 5.5)
Schritt 2: Umwandlung in Integer-Zahl	(Ergebnis 5)
Schritt 3: Umwandlung in FFP-Zahl	(Ergebnis 5)
Schritt 4: Multiplikation mit 2	(Ergebnis 10)
Schritt 5: Vergleich mit Ursprungszahl.	

Beide sind verschieden, daher ist 11 eine ungerade Zahl.

Die folgende Routine ist daher bei Verdacht auf eine negative Basis immer der eigentlichen Potenzierungsroutine vorauszustellen:

```
move.l #basis,d0     ;Basis holen (FFP-Format)
move.l #0,d1         ;kleiner null
CALLFFP SPCmp
cmp.l #1,d0
bne ok               ;nein, alles klar
move.l #2,d0
CALLFFP SPFlt        ;Zahl 2 in Fliesskomma
move.l d0,d1
move.l #exp,d0       ;Exponenten holen
CALLFFP SPDiv        ;durch 2 teilen
move.l d0,d2         ;merken
CALLFFP SPFix        ;in Integer wandeln
CALLFFP SPFlt        ;wieder in Fliesskomma
move.l d2,d1
CALLFFP SPCmp        ;beide Zahlen gleich?
cmp.l #0,d0          ;ja, dann gerade Zahl
beq gerade
move.l #$80,d2       ;Negativ-Bit bereithalten
jmp ungerade

gerade  move.l #0,d2  ;Kein Negativ-Flag
ungerade....
```

Nun können wir die fehlerhafte Potenzierungroutine aufrufen:

```
move.l #basis,d0      ;Basis im FFP-Format
move.l #exp,d1        ;Exponent im FFP-Format
CALLMATHTRANS SPPow   ;Potenzieren
```

Ergebnis in D0: `basis`^{exp} `im FFP-Format`

Als letztes müssen wir nun noch ein etwaiges Negativ-Flag einblenden. Dies kann mit dem logischen OR-Befehl geschehen:

```
or.l d2,d0            ;Flag einblenden
```

Je nachdem, ob der Inhalt des Registers 0 oder $80 war, passiert gar nichts oder wird das siebte Bit zwangsgesetzt. Es ist bedauerlich, daß man so etwas auf einem Rechner wie dem Amiga erleben muß. Meine Herren von Commodore, das können Sie doch besser!

n) Ziehen der Quadratwurzel einer FFP-Zahl: **SPSqrt**

Der Aufruf gleicht dem der transzendenten Funktionen, das Argument muß jedoch kein Winkel im Bogenmaß, sondern eine positive FFP-Zahl sein, z.B. +9:

```
move.l #$90000044,d0      ;9 im FFP-Format
CALLMATHTRANS SPSqrt
```

Ergebnis in D0: `$C0000042   ;3 im FFP-Format`

o) Umwandlung einer FFP- in eine IEEE-Zahl: **SPTiee**
Mit dieser Funktion ist es möglich, eine FFP-Zahl in das Format mit doppelter Genauigkeit umzuwandeln. Natürlich kann das Ergebnis der Umwandlung aber nicht genauer sein als die ursprüngliche FFP-Zahl selbst. Wir wollen einmal die Umwandlung der Zahl 1 durchführen:

```
move.l #$80000041,d0        ;FFP-Format
CALLMATHTRANS SPTieee
```

Ergebnis in `D0: $3FF00000    ;IEEE-Format`
`D1: $00000000`

p) Umwandlung einer IEEE- in eine FFP-Zahl: **SPFieee**
Diese Umwandlung ist nicht immer unkritisch, da der mögliche Wertebereich der IEEE-Zahl (12-Bit-Exponent) den der FFP-Zahl (7-Bit-Exponent) übersteigt. Zudem gehen 28 Bit an Genauigkeit verloren. Als Beispiel soll uns die Zahl 3 dienen:

```
move.l #$40080000,d0        ;IEEE-Format lt. Kapitel 3.3
move.l #$00000000,d1
CALLMATHTRANS SPFieee
```

Ergebnis in `in D0: $C0000042    ;FFP-Format`

3.5 Rechnen mit der MATHIEEEDOUBBAS-Library

Diese Library stellt die Rechenoperationen für die IEEE-Zahlen zur Verfügung. Da es sich um haargenau die gleichen Funktionen handelt, die wir bereits für die FFP-Zahlen vorgestellt haben, möchte ich mich darauf beschränken, Ihnen die Aufruf-Syntax der einzelnen Makros vorzustellen. Eine IEEE-Zahl muß grundsätzlich in den Registern D0 (Langwort 1) und D1 (Langwort 2) bereitgestellt werden. In diesen Registern werden auch die Ergebnisse der Rechnungen übergeben. Bei den Funktionen, die zwei Übergabeparameter benötigen (Addition, Vergleich, etc.) muß die zweite IEEE-Zahl in den Registern D2 und D3 zur Verfügung stehen. Die Library muß vor Aufruf der nun folgenden Funktionen durch Laden des Include-Files math/mathieeedoubbas*/_lib.i zugänglich gemacht werden.

a) CALLIEEEDOUB IEEEDPFix : Wandlung in ein vorzeichenbehaftetes Langwort
b) CALLIEEEDOUB IEEEDPFlt : Wandelt Langwort in IEEE-Zahl um
c) CALLIEEEDOUB IEEEDPCmp : Vergleich (Auswertung wie bei FFP)
d) CALLIEEEDOUB IEEEDPTst : Vorzeichentest (Auswertung wie bei FFP)
e) CALLIEEEDOUB IEEEDPAbs : Absolutwert (Betrag) einer Zahl
f) CALLIEEEDOUB IEEEDPNeg : Negatbildung einer IEEE-Zahl
g) CALLIEEEDOUB IEEEDPAdd : Addition zweier Zahlen
h) CALLIEEEDOUB IEEEDPSub : Subtraktion (D2/D3 von D0/D1)
i) CALLIEEEDOUB IEEEDPMul : Multiplikation zweier IEEE-Zahlen
j) CALLIEEEDOUB IEEEDPDiv : Division (D2/D3 durch D0/D1)

3.6 Umwandlung eines Strings in eine FFP-Fließkommazahl

Das Problem bei der Fließkommarechnung besteht darin, daß man eine Zahl in diesem Format normalerweise nicht benutzt. Man kann dem Anwender daher kaum zumuten, eine Eingabe im FFP-Format zu tätigen. Wie wir in Kapitel 2 gesehen haben, bieten sich vielmehr die String-Gadgets für Eingaben aller Art an, auch für die von Fließkommazahlen.

Die C64-Umsteiger werden hierin wohl kaum ein Problem sehen, da dieser Computer mit einer ROM-Routine ausgestattet ist, die die Umwandlung problemlos vollzieht. Leider wird aber oft übersehen, daß sich die Routine nicht im Betriebssystem (wo sie auch nichts zu suchen hätte), sondern im Basic-Interpreter des ROMs befindet. Beim Amiga sind Betriebssystem und Basic jedoch streng getrennt, was man daran erkennen kann, daß das Basic erst nachgeladen werden muß.

Offensichtlich erweist sich in diesem Punkt ein riesiger Vorteil der Assemblerprogrammierung, nämlich daß die Programme sofort ohne Interpreter ausführbar sind, als Bumerang. Wir stehen nämlich ohne Umwandlungsroutine da. Davon abgesehen, hätten wir aber auch bei nachgeladenem Interpreter keine Vorteile, da dieser in C geschrieben ist und wir somit keinen reinen Assembler-Code, sondern ein Kompilat zu analysieren hätten.

Offenbar müssen wir eine solche Routine selbständig entwickeln. Zunächst hatte ich vor, die C64-Routine an den Amiga anzupassen, da sie sehr universell eingesetzt werden kann (z.B. erlaubt sie auch Eingaben im Exponentialformat).

Da diese Routine aber relativ kompliziert arbeitet, habe ich es vorgezogen, eine einfach zu verstehende zu entwickeln, die allerdings keine Eingaben im Exponentialformat zuläßt. Sie können aber beliebig lange Dezimalstrings verwenden. Bei diesen sind neben den Zahlen auch der Punkt und das Minuszeichen erlaubt. Das Pluszeichen ist verboten, da es aber sowieso überflüssig ist und normalerweise nicht mit eingegeben wird, bedeutet dieses keine Einschränkung.

Die Texteingabe soll in einem Stringgadget erfolgen, da einerseits so dem Anwender eine komfortable Eingabemöglichkeit zur Verfügung steht und andererseits die Länge der Eingabe von Intuition automatisch in die String-Info-Struktur ab der Adresse 16 eingetragen wird (siehe Kapitel 2.4). Bevor ich weitere Erläuterungen gebe, sehen wir uns den relativ kurzen Quelltext einmal an:

Übergabeparameter: A1 Langwort mit Zeiger auf String-Info-Struktur
A2 Zeiger auf Stringpuffer des Gadgets

```
stringfloat       clr.l float
                  clr.w EXP
                  clr.l d0
                  clr.l d3
                move.w 16(a1),d5         ;Länge Text holen
```

```
                move.b (a2),d0          ;erstes Zeichen
                cmp.b #"-",d0           ;Minuszeichen?
                bne plus
* Negative Zahl

                move.b #128,EXP         ;Negativ-Flag setzen
                move.w d5,d0
                move.l a2,a0            ;und Rest-String
strloop1        move.b 1(a0),(a0)+
                subq #1,d0              ;um eine Position nach
                bne strloop1
                subq #1,d5              ;links schieben

plus            clr.l d0                ;Position
                move.l a2,a0
strloop2        move.b (a0)+,d3         ;Zeichen holen
                cmp.b #".",d3
                beq found               ;Punkt gefunden
                addq #1,d0
                cmp.w d0,d5             ;alles durchsucht
                beq nopoint
                jmp strloop2

found           subq #1,d0
                move.l d0,d4            ;höchster Exponent
                addq #1,d0
                move.l a2,a0            ;Punkt aus String
strloop3        move.l a0,d1
                add.l d0,d1             ;eleminieren
                move.l d1,a6
                move.b 1(a6),(a6)
                addq #1,d0
                cmp.w d0,d5
                bne strloop3
                subq #1,d5
                jmp weiter1

nopoint         move.l d5,d4
                subq #1,d4              ;höchster Exponent

weiter1         clr.l d6                ;Position
                move.l a2,a5

strloop4        move.l d4,d0            ;Exponenten holen
                CALLFFP SPFlt           ;in Fließkomma
                move.l d0,d1
                move.l stellentab,d0    ;10 in Fließkomma
```

```
                CALLMATHTRANS SPPow     ;10 hoch x rechnen
                move.l a5,d1            ;Faktor holen (ASCII)
                add.l d6,d1
                move.l d1,a6
                clr.l d1
                move.b (a6),d1
                sub.b #"0",d1           ;minus 48 ("0") =
                move.l d0,d7            ;Zahl in Fließkomma
                move.l d1,d0            ;mit 10 hoch x
                CALLFFP SPFlt
                move.l d7,d1
                CALLFFP SPMul           ;Multiplizieren
                move.l float,d1
                CALLFFP SPAdd           ;zu bisheriger Summe
                move.l d0,float         ;addieren
                sub.l #1,d4             ;Exponenten verringern
                addq #1,d6              ;Position erhöhen
                cmp.l d6,d5             ;fertig?
                bne strloop4
                move.l float,d0         ;Negativ-Flag einblenden
                or.b EXP,d0
                rts

float           dc.l 0
EXP             dc.w 0
```

Als erstes wird die Länge der Eingabe in das Register D5 geholt. Dann wird das erste Zeichen des Textes geholt und mit dem Minuszeichen verglichen. Stimmen beide überein, wird ein Negativ-Flag in der Variablen EXP gesetzt. Dazu werden für die weitere Analyse alle noch folgenden Zeichen um eine Position nach links verschoben und die Textlänge um eins verringert. Damit wurde das Minuszeichen eliminiert, so daß im weiteren Verlauf kein Unterschied mehr zwischen positiven und negativen Zahlen gemacht werden muß.

Nun befinden wir uns beim Label »Plus«. Jetzt folgt der Teil der Routine, welcher den String nach einem eventuell gesetzten Dezimalpunkt untersucht. Wird ein solcher gefunden, passiert folgendes: Der höchste Exponent der eingegebenen Zahl wird als Position des gefundenen Punktes minus eins definiert: Dies ist korrekt, wie folgendes Beispiel belegt:

$$102315{,}123 = 1*10^5 + 0*10^4 + 2*10^3 + 3*10^2 + 1*10^1 + 5*10^0 + 1*10^{-1} + 2*10^{-2} + 3*10^{-3}$$

Punktposition 6, höchster vorkommender Exponent = 5

Des weiteren wird der Punkt aus dem String eliminiert, indem alle weiter rechts stehenden Zeichen um eine Position nach links verschoben werden und die Stringlänge nochmals um eins verringert wird.

Falls kein Punkt gefunden werden konnte, wird der höchste Exponent als Stringlänge minus eins festgesetzt. Dies leuchtet sofort ein, wenn man bedenkt, daß das letzte Zeichen die Wertigkeit 100 einnimmt. Um nochmal alles bislang Geschehene zu verdeutlichen, möchte ich Ihnen zeigen, was bei dem Label »weiter1« z.B. aus dem String -5421.12 geworden ist:

Vor Aufruf der Routine:	"-5421.12" (Länge: 8)
Nach Vorzeichenanalyse:	"5421.12" (Länge: 7)
Nach Punktanalyse:	"542112" (Länge: 6)

Nun haben wir unseren String so vorbereitet, daß wir ihn in einer Schleife verarbeiten können: Das Verfahren hierzu ist wirklich einfach zu verstehen. Zunächst nehmen wir den gerade bestimmten, höchsten Exponenten und wandeln ihn ins FFP-Format um. Die erste Zahl des Strings sagt uns, wie oft wir diesen Exponenten addieren müssen: Da das Zeichen im ASCII-Code gespeichert ist, müssen wir von ihm die Zahl 48 abziehen, weil die Zahlen 0–9 die ASCII-Codes 48 bis 57 belegen.

Nach der ASCII-Umwandlung können wir diese Zahl mit dem Wert unseres Exponenten multiplizieren und haben damit den ersten FFP-Ausdruck gewonnen (in unserem Beispiel die 5*103).

Als nächstes ist das zweite Zeichen an der Reihe: Da es um den Faktor 10 niederwertiger ist als das erste, muß der Exponent um eins verringert werden. Anschließend wird wiederum der Faktor herausanalysiert und mit dem Exponentenprodukt multipliziert. Dieses Ergebnis wird zu dem der ersten Ziffer hinzuaddiert. Sie können sich nun denken, wie es weitergeht: Bis zum letzten Zeichen wird jeweils der Exponent um eins verringert und das entstandene Produkt aus Faktor und Exponent zu der bisherigen Summe hinzuaddiert. Sie sehen auch, daß es sinnvoll war, den Dezimalpunkt zu entfernen, da wir so einfach den String chronologisch von links nach rechts abarbeiten können. Zu guter Letzt müssen wir nur noch das Negativ-Flag in unser Ergebnis im Register D0 einblenden.

Diese Routine ist wesentlich kürzer als die entsprechende im C64-ROM und ist vor allen Dingen leicht zu verstehen. Sie bietet sich dazu an, von Ihnen in eigenen Programmen verwendet zu werden, da die Länge des Strings sowie die Größe der eingegebenen Zahl (fast) unbegrenzt sind.

3.7 Die Bildschirmausgabe einer FFP-Zahl

Nachdem wir unsere Zahl nun im FFP-Format vorliegen haben, können wir nach Herzenslust mit ihr herumrechnen, indem wir die Library-Funktionen benutzen. Irgendwann wird uns der Spaß aber vergangen sein, sei es, weil unser Amiga eine Ruhepause benötigt oder weil unser Ergebnis durch die Ungenauigkeit der Rechenroutinen nur noch am Rande an das gewünschte erinnert. Zu diesem Zeitpunkt taucht dann der Wunsch auf, die Zahl auf dem Bildschirm ausgeben zu können.

Leider ist Intuition aber nur dazu in der Lage, Strings auszugeben, wie wir es in Kapitel 2 demonstriert haben. Der einzige, der uns helfen könnte, wäre der Basicinterpreter oder ein C-Compiler. Da wir aber froh sind, ohne diese auszukommen, werden wir auch für die Ausgabe eine eigene Routine entwerfen. Wenn man sich diejenige des C64-Basicinterpreters ansieht, stellt man fest, daß sie noch komplizierter und länger als die Eingaberoutine ist. Man kann also gut darauf verzichten, sie auf den Amiga zu übertragen.

Das eigentliche Problem besteht nicht in der Bildschirmausgabe, sondern in dem Verfahren, eine FFP-Zahl in einen String umzuwandeln. Um eine recht einfache Routine entwerfen zu können, mußten mehrere Einschränkungen in Kauf genommen werden:

- Keine Ausgabe im Exponentialformat
- Festgelegte Zahl der maximalen Vorkommastellen
- Festgelegte Zahl der maximalen Nachkommastellen

Letztere Einschränkungen sind in der Praxis aber nicht bedeutsam, da man die maximal zulässigen Stellen ja groß genug dimensionieren kann. Unsere Routine besitzt aber auch einen nicht zu unterschätzenden Vorteil: Sie generiert nämlich rechtsbündige Strings, so daß die Ausgabe der Zahlen 2.3 und 1124.67 nicht, wie in Basic und C, als

```
 62.3
1124.67
```

erfolgen würde, sondern rechtsbündig als

```
   2.3
1124.67
```

Wie dies genau geschieht, werden wir gleich sehen. Hier nun der Quelltext unserer Routine:

Übergabeparamter: RegisterD3 FFP-Zahl

```
          move.l #textstring1,a0
          move.b #" ",(a0)+           ;Platz für Vorzeichen
          move.l #30,d0               ;Zähler für Nullen
clearloop move.b #"0",(a0)+
          subq #1,d0
          bne clearloop
          move.l #textstaring,a0
          move.b #".",20(a0)          ;Dezimalpunkt setzen

          move.l #textstring1,a0
          cmp.b #$80,d3               ;negative Zahl?
          bmi positiv
          move.b #"-",(a0)            ;Ja, Minuszeichen einfügen
          sub.b #$80,d3
positiv   move.l #1,d2                ;Durchgang
```

```
         move.l #zehnhochxtab,d6     ;Tabelle mit Potenzen
         move.l #30,d5               ;Anzahl der Nachkommastellen
w3       move.l #19,d0               ;höchster Exponent
         sub.l d2,d0
         add.l #10,d0                ;Offset
         lsl.l #2,d0                 ;mal 4 wegen Langwort
         add.l d6,d0                 ;plus Basis der Tabelle
         move.l d0,a5
         move.l (a5),d0              ;Wert holen
         move.l d0,d7
w2       move.l d7,d0
         move.l d3,d1                ;mit Zahl vergleichen
         CALLFFP SPCmp
         cmp.l #-1,d0
         beq next                    ;10 hoch x größer
         move.l #textstring1,d0
         add.l d2,d0
         cmp.l #20,d2
         blt w1
         addq #1,d0                  ;Wert der Position im
w1       move.l d0,a0                ;String um eins
         clr.l d0
         move.b (a0),d0              ;erhöhen
         addq #1,d0
         move.b d0,(a0)
         move.l d3,d0
         move.l d7,d1                ;Zahl verringern um
         CALLFFP SPSub
         move.l d0,d3                ;10 hoch x
         jmp w2

next     addq #1,d2                  ;Zähler erhöhen
         cmp.l d5,d2
         bne w3

*Überflüssige Nachkommastellen abschneiden

         move.l #30,d0
p1       move.l #textstring1,d1
         add.l d0,d1
         move.l d1,a0                ;wenn "0" gefunden,
         move.b (a0),d2
         cmp.b #"0",d2               ;eleminieren
         bne nonull
         subq #1,d0
         cmp.l #20,d0
         bne p1
```

```
         move.b #0,-(a0)             ;keine Nachkommastellen
         jmp p4
nonull   move.b #0,1(a0)

* Überflüssige Nullen am Anfang durch Leerzeichen ersetzen

p4       move.l #textstring1,d1
         add.l #1,d1
         move.l d1,a0
p2       move.b (a0)+,d3              ;Zeichen holen
         cmp.b #"0",d3                ;"0"?
         bne pfertig                  ;nein, fertig
         move.b (a0),d0
         cmp.b #".",d0                ;"."?
         beq pfertig                  ;ja, fertig
         cmp.b #0,d0
         beq pfertig                  ;Leerzeichen einfügen
         move.b -2(a0),-1(a0)
         move.b #" ",-2(a0)
        jmp p2

pfertig rts

* Potenzen im Fließkommaformat

  even

zehnhochxtab dc.b $DB,$E6,$FE,$1F ;10 hoch -10
             dc.b $89,$70,$5F,$23 ;10 hoch -9
             dc.b $AB,$CC,$77,$26 ;10 hoch -8
             dc.b $D6,$BF,$46,$29 ;10 hoch -7
             dc.b $86,$37,$BD,$2D ;10 hoch -6
             dc.b $A7,$C5,$AC,$30 ;10 hoch -5
             dc.b $D1,$B7,$17,$33 ;10 hoch -4
             dc.b $83,$12,$72,$37 ;10 hoch -3
             dc.b $A3,$D7,$0A,$3A ;10 hoch -2
             dc.b $CC,$CC,$CC,$3D ;10 hoch -1
             dc.b $80,$00,$00,$41 ;10 hoch 0
             dc.b $A0,$00,$00,$44 ;10 hoch 1
             dc.b $C8,$00,$00,$47 ;10 hoch 2
             dc.b $FA,$00,$00,$4A ;10 hoch 3
             dc.b $9C,$40,$00,$4E ;10 hoch 4
             dc.b $C3,$50,$00,$51 ;10 hoch 5
             dc.b $F4,$24,$00,$54 ;10 hoch 6
             dc.b $98,$96,$80,$58 ;10 hoch 7
             dc.b $BE,$BC,$20,$5B ;10 hoch 8
             dc.b $EE,$6B,$28,$5E ;10 hoch 9
```

```
            dc.b $95,$02,$F9,$62 ;10 hoch 10
            dc.b $BA,$43,$B7,$65 ;10 hoch 11
            dc.b $E8,$D4,$95,$68 ;10 hoch 12
            dc.b $91,$84,$D1,$6C ;10 hoch 13
            dc.b $B5,$E5,$F7,$6F ;10 hoch 14
            dc.b $E3,$5F,$83,$72 ;10 hoch 15
            dc.b $8E,$1B,$A7,$76 ;10 hoch 16
            dc.b $B1,$A2,$9B,$79 ;10 hoch 17
            dc.b $DE,$0B,$50,$7C ;10 hoch 18

textstringl dc.b " 0000000000000000000.0000000000",0
```

Keine Angst, liebe Leser, das Programm sieht komplizierter aus als es wirklich ist. Die Routine ist für Zahlen von 10^{-10} bis 10^{18} geeignet. Dieser Bereich dürfte wohl für 99,9% aller Anwendungsfälle ausreichen: Selbst wenn man auf die (besonders intelligente) Idee kommen würde, das Gewicht eines Supertankers (300000 Tonnen) in Milligramm aufzuzeigen, würde man »nur« die Zahl 3 * 1014 erreichen. Umgedreht haben wir ja festgestellt, daß die FFP-Zahlen nur eine Genauigkeit von sechs Stellen hinter dem Komma haben, so daß es sowieso schon übertrieben ist, mit zehn Nachkommastellen zu rechnen.

Zunächst wird unser Stringpuffer, den wir nachher auf den Bildschirm ausgeben wollen, initialisiert. Dazu wird zunächst ein Leerzeichen für das Vorzeichen, dann insgesamt 30 Nullen im ASCII-Code eingetragen. Zum Schluß wird noch der Dezimalpunkt gesetzt. Dieser String ist bereits technisch gesehen ausgabefähig, auch wenn ihm noch jeder sinnvolle Inhalt fehlt.

Nach der Initialisierungsroutine wird die FFP-Zahl zunächst auf ihr Vorzeichen überprüft. Dazu wird das Byte, das den Exponenten sowie das Negativ-Bit enthält, mit dem Wert $80 verglichen. Falls das Negativ-Bit gesetzt ist, wird das gleichnamige Flag des Prozessors gelöscht. Daraufhin wird ein Minuszeichen an die erste Position unseres Strings geschrieben. Falls die FFP-Zahl positiv ist, kann das Leerzeichen der Initialisierung bestehen bleiben.

Nun beginnt die eigentliche Umwandlung: Zunächst wird der Wert der höchsten vorkommenden Zehnerpotenz (18) aus einer Tabelle geholt. Die Zehnerpotenzen wurden deshalb in einer Tabelle abgelegt, um die Zeit der Umwandlung ins Fließkommaformat zu sparen, die sich sehr nachhaltig ausgewirkt hätte. Dazu wird der jeweilige Durchgang (1–29) von dem Startwert 19 in Register D5 abgezogen, was die Zahlen 18 bis –10 ergibt. Hierbei handelt es sich um die vorkommenden Zehnerexponenten in absteigender Reihenfolge. Nach Addieren eines Offsets (10) und einer Multiplikation mit 4 durch zweimaliges Linksschieben kann dieser Wert als Offset für die Tabelle mit den Potenzen benutzt werden. So wird im ersten Durchgang der Wert 10^{18} ausgelesen, im nächsten dann 10^{17} usw. Der Trick besteht jetzt darin, die FFP-Zahl mit dieser Potenz zu vergleichen. Falls die FFP-Zahl kleiner ist, kommt offenbar diese Zehnerpotenz nicht vor und der Durchgang kann beendet werden und anschließend mit der nächstniedrigeren Potenz fortgefahren

werden. Falls jedoch die FFP-Zahl größer oder gleich ist, wird zum einen die entsprechende Stelle im String um eins erhöht und dann die FFP-Zahl genau um diese Potenz verringert. Anschließend wird wieder verglichen und solange weitergemacht, bis dann endlich die FFP-Zahl kleiner wird als die Potenz. Auf diese Weise wird die entsprechende Position im String immer weiter um eins erhöht. Weil dort anfangs der ASCII-Code für die Zahl Null zu finden war, ist z.B. nach sechsmaligem Durchlauf dort der Code für die Zahl 6 zu finden. Auf diese Weise werden auch die Stellen der Potenzen 10^{17} bis 10^{-11} durchlaufen. Da maximal pro Potenz neun Durchgänge erforderlich sein können, muß man im schlimmsten Fall mit 29*9=261 Durchgängen rechnen. Ein Sonderfall tritt an der Stringposition 20 auf: Da sich hier der Dezimalpunkt befindet, muß man den Positionszeiger an dieser Stelle um eins erhöhen.

Nachdem alle 29 Durchläufe beendet wurden, ist der eigentliche String bereits erzeugt worden. Was nun folgt, sind kosmetische Korrekturen. Zum einen ist es üblich, überflüssige Nachkommastellen abzuschneiden. Wer möchte schon gerne die Ausgabe

21.2300000000

sehen? Sie auch nicht, dann sind wir uns ja einig. Hierzu werden die Nachkommastellen rückwärts überprüft, ob eine Null im ASCII-Code vorhanden ist. Offenbar ist das Ende des Strings dann gekommen, wenn die erste Null von rechts gefunden wurde. Ein Fehler wäre es hingegen, die Überprüfung von links, also vom Dezimalpunkt aus zu starten: Die erste hier gefundene Null würde nicht zwangsläufig das Stringende bedeuten, z.B.

12.123012

An das Ende des Strings wird eine Null (nicht deren ASCII-Code!) gesetzt, was für die String-Ausgabe-Routine das Zeichen ist, daß der String beendet ist. Nun müssen wir uns mit den Vorkommastellen beschäftigen. Auch hier wollen (müssen) wir die überflüssigen Nullen eliminieren, sonst hätten wir möglicherweise eines Tages eine Ausgabe wie

-000000000001123458.12

zu bestaunen. Dazu wird von links aus bis zum Punkt geprüft, wie lange Nullen in dem String enthalten sind. Wird der ASCII-Code einer Null nicht gefunden, ist auf jeden Fall der Anfang des »sinnvollen« Strings gekommen, weshalb man zum Ende der Routine springen kann. Wird jedoch eine Null gefunden, ist Vorsicht angesagt. Grundsätzlich, müßte man denken, ist es erforderlich, die Null durch ein Leerzeichen zu ersetzen, um die rechtsbündige Ausgabe zu erreichen. Wenn man dann noch ein etwaiges Minuszeichen um eine Stelle nach rechts verschiebt, um dies direkt vor den String zu positionieren, scheint alles erledigt zu sein und man könnte mit der Prüfung beim nächsten Zeichen fortfahren.

Leider gibt es aber zwei Ausnahmen, welche die Nullen unbedingt vor dem Verschwinden retten müssen: Zum einen dann, wenn es sich bei der gefundenen Null um den Code direkt vor dem Dezimalpunkt handelt, da wir uns eine schöne Ausgabe wie

0.125

und nicht die schreckliche Computerschreibweise

.125

programmieren wollen. Der andere Fall ist noch wichtiger: Wenn unsere FFP-Zahl nämlich eine schlichte Null darstellte. In diesem Fall besteht der erzeugte String nur aus einem Zeichen, nämlich der Null. Wenn wir diese durch ein Leerzeichen ersetzen würden, könnte man gar nichts mehr sehen!

Sie sehen, liebe Leser, daß man auch mit relativ einfachen Mitteln zum Ziel kommt. Den erzeugten String können Sie nun bequem mit der Textausgabe-Funktion von Intuition auf den Bildschirm ausgeben. Genausogut können Sie ihn aber auch zum Drucker schicken oder auf einer Diskette verewigen, da diese Ausgaberoutinen dasselbe Format erwarten. Damit haben wir das größte Problem bei der Fließkommarechnung in Maschinensprache gelöst und können uns nun beruhigt unserem erstem Beispielprogramm zuwenden.

3.8 Programmierung einer Luxus-Wertetabelle

Wenn es um Fließkommarechnung geht, bietet sich eine Wertetabelle als Beispielprogramm geradezu an. Wir wollen uns aber nicht damit begnügen, Minimal-Software zu erstellen, sondern stellen folgende Ansprüche :

- **Darstellung von sämtlichen Polynomen der Form**

$$y = \frac{a1*x^{e1}+a2*x^{e2}+a3*x^{e3}+a4*x^{e4}+a5*x^{e5}+a6*x^{e6}+a7*x^{e7}+a8*x^{e8}}{b1*x^{n1}+b2*x^{n2}+b3*x^{n3}+b4*x^{n4}+b5*x^{n5}+b6*x^{n6}+b7*x^{n7}+b8*x^{n8}}$$

  mit a1..a8, b1..b8 : Faktoren von Zähler und Nenner
  e1..e8, n1..n8 : Exponenten von Zähler und Nenner

  Durch ein solches Polynom können Sie auch z.B. die transzendeneten Funktionen mittels einer Potenzreihe darstellen (Bild 3.1).

- **Auswahl des Ausgabebereiches und der Genauigkeit**
  Sie können untere, obere Grenze und Schrittweite zwischen zwei betrachtenden x-Werten genauso frei festlegen, wie die Anzahl der Nachkommastellen, auf die Ihre Funktionsergebnisse gerundet werden (1–3).

- **Maus- und Menüsteuerung**
  Die Eingabe der Faktoren, Exponenten und Parametern wird durch Gadgets vollzogen, die Auswahl der einzelnen Menüpunkte geschieht in einem Pull-down-Menü.

- **Wahlweise Bildschirm- oder Druckerausgabe der Funktionswerte**
  Die Bildschirmausgabe erfolgt tabellarisch und rechtsbündig. Es werden nur immer so viele Funktionswerte ausgegeben, bis der Bildschirm gefüllt ist. Die weiteren (vorher-

gehenden) Werte können Sie durch Abwärts- (Aufwärts) Scrolling erhalten. Bei der Druckerausgabe wird eine formatierte Liste mit Kopf und eingerahmten Funktionswerten ausgegeben.

- **Grafische Funktionsdarstellung mit Wahl des y-Maßstabes**
  Sie können wahlweise nur Zähler oder Nenner sowie die komplette Funktion plotten. Dabei haben Sie die Auswahl zwischen dem Maßstab 1:1, was bedeutet, daß die Funktion unverzerrt dargestellt wird. Allerdings ist es möglich, daß nicht alle Funktionswerte dargestellt werden, nämlich dann, wenn Sie den festgelegten x-Bereich überschreiten. Um einen Überblick zu gewinnen, bietet sich deshalb der Modus »Anpassung« an, in dem der y-Maßstab soweit verkleinert wird, daß garantiert alle Funktionswerte dargestellt werden können. Die Grafik ist dann aber kein Abbild der wirklichen Funktion mehr, sondern wird gequetscht angezeigt.

Sie sehen, daß unsere Wertetabelle für ein in einem Buch veröffentlichtes Programm außergewöhnliche Features aufweist und vor allen Dingen die Sachen in der Praxis benutzt, welche wir vorher theoretisch besprochen haben. Von Gadgets, Screens, Windows über Menüs, den Message-Port und Fließkommarechnung ist alles vorhanden. Etwas zu früh kommen die Grafikroutinen, die wir ausführlich im nächsten Kapitel behandeln werden. Was aber wäre eine Luxus-Wertetabelle ohne Funktionsplotter?

Bevor wir nun zu der Besprechung des Quelltextes übergehen wollen, starten Sie doch bitte einmal das Programm mit dem Namen »Wertetabelle« vom CLI oder von der Workbench. Nachdem die Floppy anläuft, um die Mathtrans-Library einzuladen, erscheint das Startwindow. Sie können jetzt einen Menüpunkt auswählen. Bitte wählen Sie in dem Menü »Eingabe« den Punkt »Zähler« oder »Nenner« aus. Es erscheint nun ein weiteres Window mit jeweils acht untereinanderstehenden Stringgadgets auf der linken und rechten Seite des Windows, dazwischen steht der Buchstabe x. Das heißt nichts anderes, als daß auf der linken Seite die Faktoren und auf der rechten die Exponenten zu sehen sind. Wenn Sie nicht alle acht Potenzen benutzen möchten, müssen Sie die Faktoren der überflüssigen Potenzen zu Null machen. Ein Exponent von Null bedeutet, daß der Wert des Faktors konstant ist und nicht von x abhängt (z.B. 2*x0=2).

Nachdem Sie die Eingabe beendet haben, wählen Sie bitte den Menüpunkt »Parameter« im gleichen Menü an. Es erscheinen in einem weiteren Window drei Stringgadgets und ein Proportionalgadet. Durch die Gadgets »Obere Grenze« und »Untere Grenze« wird der Bereich festgelegt, in dem die Funktionswerte berechnet werden, durch die »Schrittweite« der Abstand zwischen je zwei berechneten x-Werten. Das Proportionalgadget legt die Anzahl der Nachkommastellen fest, auf die x- und y-Werte gerundet werden. Dazu klicken Sie das Gadget an und verschieben den Pfeil unter die Zahl, die der gewünschten Anzahl der Nachkommastellen entspricht.

In dem ersten Menüpunkt des zweiten Menüs (»Options«) können Sie zwischen der Bildschirm- und Druckerausgabe wählen. Falls Sie den Ausdruck gewählt haben, versuchen Sie einmal, während des Druckes einen weiteren Menüpunkt anzuwählen. Dieses wird scheitern, da alle Menüpunkte unwählbar gemacht wurden, also punktiert dargestellt werden.

In dem zweiten Menüpunkt legen Sie den Maßstab für die grafische Ausgabe fest, der angewählte Maßstab wird immer abgehakt dargestellt. Der letzte Menüpunkt schließlich beendet das Programm.

$\mathtt{COS(X)} = 1-0.5*x^2+0.041667*x^4-0.001389*x^6+0.0000248*x^8-0.0000002756*x^{10}$

$\mathtt{COSH(X)} = 1+0.5*x^2+0.041667*x^4+0.001389*x^6+0.0000248*x^8+0.0000002756*x^{10}$

$\mathtt{SIN(X)} = x-0.1667*x^3+0.00833*x^5-0.0001984*x^7+0.000002756*x^9-0.0000000251*x^{11}$

$\mathtt{SINH(X)} = x+0.1667*x^3+0.00833*x5+0.0001984*x^7+0.000002756*x^9+0.0000000251*x^{11}$

$\mathtt{TAN(X)} = \mathtt{SIN(X)/COS(X)}$

$\mathtt{COT(X)} = \mathtt{COS(X)/SIN(X)}$

$\mathtt{EXP(X)} = 1+x+0.5*x^2+0.1667*x^3+0.041667*x^4+0.00833*x^5+0.001389*x^6+0.000198*x^7$

$\mathtt{LOG\ (X)} = \mathtt{LOG}(X_0) + (1/X_0)*(X-X_0) - (1/(2*X_0^2))*(X-X_0)^2 + (1/(3*X_0^2))*(X-X_0)^3$
$- (1/(4*X_0^4))*(X-X_0)^4 + (1/(5*X_0^5))*(X-X_0)^5 - (1/(6*X_0^6))*(X-X_0)^6$

(gilt in der Umgebung von X_0)

Bild 3.1: Potenzreihen einiger Funktionen

In dem letzten Menü »Grafik« können Sie sich wahlweise Zähler, Nenner oder die Funktion plotten lassen. Dazu wird ein Screen in hoher Auflösung (640 Punkte in x-Richtung) geöffnet. Danach müssen Sie ca. 20 Sekunden warten, bis der Plot erscheint. Diese Zeit mutet für Assembler-Verhältnisse als geradezu gigantisch an. Bitte bedenken Sie aber, wie viele Rechenoperationen durchgeführt werden müssen: Pro Grafikpunkt sind jeweils für Zähler und Nenner acht Potenzierungen, acht Multiplikationen und acht Additionen notwendig. Dazu kommt die Division Zähler durch Nenner, insgesamt also 25 Fließkommarechnungen. Da wir 640 Funktionswerte berechnen müssen, sind insgesamt 16000 Fließkommarechnungen erforderlich. Die Umsteiger vom C64 können sich ausmalen, wie lange dieser Rechenzwerg dazu gebraucht hätte! Man könnte annehmen, daß die Maschinensprache bei Fließkommaoperationen kaum Vorteile bringt, da die gleichen Libraries benutzt werden wie vom Basic und C aus. Ich kann Ihnen aber verraten, daß wir in Assembler ca. fünfmal schneller sind als in Basic und ca. 1.5mal schneller als in C. Selbst in dem ungünstigsten Fall für die Maschinensprache, der Fließkommarechnung, stellt sich diese als unschlagbar heraus.

Die Potenzreihen weiterer Funktionen findet man im »Bronstein«, Taschenbuch der Mathematik, erschienen im Verlag Harri Deutsch unter der Nummer ISBN-3871440167.

3.8.1 Include-Files und Makrodefinitionen

HiSoft GenAmiga Assembler 1.21 "Wertetabelle"

```
            opt l-,d+,s-,n+
            plen 44
            llen 80

* INCLUDE-Files laden

      include df0:include/exec/exec_lib.i
      include df0:include/math/mathffp_lib.i
      include df0:include/intuition/intuition_lib.i
      include df0:include/math/mathtrans_lib.i
      include df0:include/graphics/graphics_lib.i
      include df0:include/libraries/dos_lib.i

* MACRO-Definitionen

* Intuition-Text ausgeben

PRINTOUT     macro
             lea \1,a1
             move.l #\2,d0
             move.l #\3,d1
             move.l windowpointer,a0
             move.l 50(a0),a0
             CALLINT PrintIText
             endm

*Struktur für Ausgabe-Text

TEXTOUT      macro
             dc.b 3,0           ;Farben
             dc.b  0            ;Farbmodus
             even
             dc.w  0            ;x-Koordinate
             dc.w  0            ;y-Koordinate
             dc.l  0            ;Zeichensatz
             dc.l  \1           ;Textpointer
             dc.l  0            ;weiterer Text
             endm

TEXTOUT1     macro
             dc.b 0,0           ;Farben
             dc.b  0            ;Farbmodus
             even
```

```
                dc.w \1          ;x-Koordinate
                dc.w \2          ;y-Koordinate
                dc.l  0          ;Zeichensatz
                dc.l \3          ;Textpointer
                dc.l  0          ;weiterer Text
                endm

*Struktur für Menü
MENUE           macro
                dc.l \1          ;nächstes Menü
                dc.w \2,\3       ;x,y des Menütitels
                dc.w 70,10       ;Breite,Höhe des Menütitels
                dc.w 1           ;Menü wählbar
                dc.l \4          ;Menütitel
                dc.l \5          ;Menüeinträge
                dc.w 0,0,0,0
                endm

*Struktur für Menüpunkte

MENUPUNKT       macro
                dc.l  \1        ;nächster Menüpunkt
                dc.w  \2,\3     ;x,y-Position
                dc.w  \4        ;Breite
                dc.w  10        ;Höhe
                dc.w  \5        ;Flag
                dc.l  \6        ;kein Ausschluß
                dc.l  \7        ;Zeiger auf Textstruktur
                dc.l  0         ;keine Zeichnung
                dc.b  \8        ;Amiga-Taste
                even
                dc.l  \9        ;Zeiger auf Untermenü
                dc.l  0
                endm

* Struktur für Windows

WINDOW          macro
                dc.w      \1    ;linke Ecke
                dc.w      \2    ;obere Ecke
                dc.w      \3    ;Breite
                dc.w      \4    ;Höhe
                dc.b      0,1   ;Farben
                dc.l      \5    ;Flags für Events
                dc.l      \9    ;Window Flags
                dc.l      \6    ;Zeiger auf erstes Gadget
                dc.l      0     ;Check Mark
```

```
             dc.l      \7    ;Windowname
\8           dc.l      0     ;Screenpointer
             dc.l      0     ;Bit Map
             dc.w     \3     ;Mindest-Breite
             dc.w     \4     ;Mindest-Höhe
             dc.w     \3     ;Maximale Breite
             dc.w     \4     ;Maximale Höhe
             dc.w     15     ;Window-Typ
             endm

* Struktur für Screens

SCREEN       macro
             dc.w      0     ;linke Ecke
             dc.w      0     ;obere Ecke
             dc.w      \1    ;Breite
             dc.w      \2    ;Höhe
             dc.w      2     ;Tiefe (Bitplanes)
             dc.b      2,3   ;Farben
             dc.w      \3    ;Modus
             dc.w     15     ;Screentyp
             dc.l      0     ;Zeichensatz
             dc.l      \4    ;Screentitel
             dc.l      0     ;Gadgets
             dc.l      0     ;Bitmap
             endm

* Struktur für Gadgets

GADGET       macro
             dc.l      \1    ;nächstes Gadget
             dc.w      \2    ;obere Ecke x
             dc.w      \3    ;obere Ecke y
             dc.w     122    ;Breite
             dc.w      10    ;Höhe
             dc.w       0    ;Flags
             dc.w       2    ;Activation-Flags
             dc.w      \4    ;Gadget-Typ:String
             dc.l      \5    ;Gadget-Border
             dc.l       0    ;kein Select-Gadget
             dc.l      \6    ;Gadget-Text
             dc.l       0    ;kein Exclude
             dc.l      \7    ;Special-Info
             dc.w      \8    ;Gadget-ID
             dc.l       0    ;User-Data
             endm
```

```
* Struktur für Eingaben bei Stringgadgets

STRINF        macro
              dc.l  \1          ;Textpuffer
              dc.l  undo        ;Undo-Puffer
              dc.w  0           ;Cursor-Position
              dc.w  15          ;Anzahl Zeichen (max)
              dc.w  0           ;Ausgabeposition Cursor
              dc.w  0           ;Position Undo-Puffer
              dc.w  0           ;Anzahl Zeichen im Textpuffer
              dc.w  0           ;Anzahl sichtbare Zeichen
              dc.w  0           ;horizontaler Offset
              dc.w  0           ;vertikaler Offset
              dc.l  0           ;Zeiger auf Rastport
              dc.l  0           ;Integer-Wert
              dc.l  0           ;Tastatur-Belegung
              endm

* Textstruktur für Gadgets

STRING        macro
              dc.b  1,0         ;Farben
              dc.b  1           ;Farbmodus
              even
              dc.w  \1,\2       ;Positionen
              dc.l  0           ;Zeichensatz
              dc.l  \3          ;Textpointer
              dc.l  \4          ;nächster Text
              endm

* Strings für Eingaben

ZAHLENSTRINGS   macro
                dc.b  \1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                dc.b  \7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
```

```
                dc.b  \8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                even
                endm

         * Fließkommazahlen

         ZAHLENFLOAT    macro
                dc.b   \1,0,0,\2
                dc.b   $80,0,0,$41
                dc.b   $80,0,0,\6
                dc.b   \3,0,0,\6
                dc.b   $80,0,0,\7
                dc.b   \4,0,0,\7
                dc.b   \3,0,0,\7
                dc.b   \5,0,0,\7
                endm

0000006E undo    dc.l  0,0,0,0          ;Undo-Puffer für Stringgadets

0000007E begin   lea ffpname,a1         ;Math-Library öffnen
00000084         clr.l d0
00000086         CALLEXEC OpenLibrary
0000008E         tst.l d0
00000090         beq fini1
00000094         move.l d0,_MathBase

0000009A         lea intname,a1         ;Intuition-Library öffnen
000000A0         clr.l d0
000000A2         CALLEXEC OpenLibrary
000000AA         tst.l d0
000000AC         beq fini2
000000B0         move.l d0,_IntuitionBase

000000B6         lea mathtransname,a1  ;Mathtrans-Library öffnen
000000BC         clr.l d0
000000BE         CALLEXEC OpenLibrary
000000C6         tst.l d0
000000C8         beq fini3
000000CC         move.l d0,_MathTransBase

000000D2         lea grafname,a1        ;Grafik-Library öffnen
000000D8         clr.l d0
000000DA         CALLEXEC OpenLibrary
000000E2         tst.l d0
000000E4         beq fini4
000000E8         move.l d0,_GfxBase
```

```
000000EE        lea dosname,a1        ;DOS-Library öffnen
000000F4        clr.l d0
000000F6        CALLEXEC OpenLibrary
000000FE        tst.l d0
00000100        beq fini5
00000104        move.l d0,_DOSBase
```

Bislang ist noch nichts weltbewegendes passiert: Am Anfang stehen die Include-Files, die wir für unsere Library-Funktionen benötigen. Eigentlich enthalten sie noch viel mehr Funktionen, ich habe aber, um Platz zu sparen, viele nachträglich gelöscht. Wenn Sie sich einmal ein Listing erstellen, werden Sie feststellen, daß die Include-Files mindestens zehn Seiten benötigen.

Innerhalb der Include-Files erkennt man die Makrodefinition, die es uns ermöglicht, die Funktion mit CALL Name aufzurufen. Uns bleibt es somit erspart, die Offsets der einzelnen Funktionen zu kennen.

Nun folgen die Makrodefinitionen: Zum großen Teil werden Sie sie aus Kapitel 2 wiedererkennen. Das Fließkomma-Makro dient dazu, die voreingestellten Zahlen von Zähler und Exponent im FFP-Format in das Programm zu integrieren. Analog hat das Makro ZAHLENSTRINGS die Aufgabe, diese voreingestellten Zahlen im String-Format für die String-Gadgets bereitzuhalten. Am Schluß der Makros steht der Undo-Puffer, den sich alle String-Gadgets teilen müssen.

Nun folgt der Programmbeginn. Zunächst werden alle benötigten Libraries geöffnet. Schlägt ein Öffnungsversuch fehl, womit nicht zu rechnen ist, werden die Label fini angesprungen, die ganz am Ende des Programms stehen. Der einzige, realistisch mögliche Fehler kann beim Öffnen der Mathtrans-Library auftreten, da diese nicht ständig im ROM präsent ist, sondern von der Diskette nachgeladen werden muß. Falls der Programmstart fehlschlägt, aber ein Anlaufen des Laufwerkes beobachtet werden kann, sollten Sie zunächst überprüfen, ob die eingelegte Diskette die Mathtrans-Library überhaupt enthält (wie z.B. die DEVPAC-Original-Diskette).

3.8.2 Auswertung des Message-Ports und der Menüs

```
0000010A        lea screen_defs,a0          ;Screen öffnen
00000110        CALLINT OpenScreen
0000011A        move.l d0,screenpointer     ;Pointer für
00000120        move.l d0,point1
00000126        move.l d0,point2            ;Windows setzen
0000012C        move.l d0,point3
00000132        move.l d0,point4
00000138        move.l d0,point5
```

```
        * menueaufbau, abfrage

0000013E        lea startwindow,a0        ;Startwindow öffnen
00000144        CALLINT OpenWindow
0000014E        move.l d0,windowpointer
00000154        bsr  print                ;Text ausgeben
00000158        move.l windowpointer,a0
0000015E        lea menu,a1
00000164        CALLINT SetMenuStrip      ;Menü setzen
0000016E        bra  wait

00000172 start  clr.l d4                  ;Menü
00000174 einl   clr.l d3                  ;Menüpunkt
00000176 einl1  clr.l d5                  ;Untermenüpunkt
00000178        move.l d3,d2              ;Menüpunkt nach Bit 5-10
0000017A        lsl.l #5,d2
0000017C einl2  move.l d5,d0              ;Untermenüpunkt nach Bit
0000017E        lsl.l #7,d0               ;11-15
00000180        lsl.l #4,d0
00000182        add.l d2,d0               ;Menüpunkt einblenden
00000184        add.l d4,d0               ;Menü einblenden
00000186        move.l windowpointer,a0   ;Menüpunkt
0000018C        CALLINT OnMenu            ;wählbar machen
00000196        addq.l #1,d5
00000198        cmpi.l #3,d5
0000019E        bne einl2                 ;max. Untermenüpunkte+1
000001A2        addq.l #1,d3              ;(prinzipiell beliebig)
000001A4        cmpi.l #4,d3
000001AA        bne einl1                 ;max. Menüpunkte+1
000001AE        addq.l #1,d4              ;(prinzipiell beliebig)
000001B0        cmpi.l #3,d4
000001B6        bne einl                  ;Menüs
                                          ;(tatsächliche Anzahl)
000001BA wait   move.l windowpointer,a0
000001C0        move.l 86(a0),a0          ;User-Port holen
000001C4        move.l a0,a5              ;Adresse retten
000001C6        move.b 15(a0),d1          ;Signal-Bit holen
000001CA        moveq #0,d0               ;Nummer in Maske
000001CC        bset d1,d0                ;wandeln
000001CE        CALLEXEC Wait             ;Auf Wiedersehen!
000001D6        move.l a5,a0
000001D8        CALLEXEC GetMsg           ;Message holen
000001E0        move.l d0,a1
000001E2        clr.l d7
000001E4        move.w $18(a1),d7         ;Menüpunkt holen
000001E8        CALLEXEC ReplyMsg         ;Message quittieren
000001F0        cmpi.w #$ffff,d7          ;Kein Menüpunkt gewählt
```

```
000001F4        beq wait
000001F8        move.w d7,d6
000001FA        lsr #7,d7
000001FC        lsr #4,d7
000001FE        lsl #2,d7                  ;Untermenü*4 in D7
00000200        move.w d6,d5
00000202        andi.l #%00000000000000000000011111100000,d6
00000208        lsr #3,d6                        ;Menüpunkt*4 in D6
0000020A        andi.l #%00000000000000000000000000011111,d5
00000210        lsl #2,d5                        ;Menü*4 in D5
00000212        move.l d5,a5
00000214        clr.l d4
00000216 ausl   clr.l d3                   ;alle Menüpunkt unwählbar
00000218 ausl1  clr.l d5
0000021A        move.l d3,d2               ;machen (s.o.)
0000021C        lsl.l #5,d2
0000021E ausl2  move.l d5,d0
00000220        lsl.l #7,d0
00000222        lsl.l #4,d0
00000224        add.l d2,d0
00000226        add.l d4,d0
00000228        move.l windowpointer,a0
0000022E        CALLINT OffMenu
00000238        addq.l #1,d5
0000023A        cmpi.l #3,d5
00000240        bne ausl2
00000244        addq.l #1,d3
00000246        cmpi.l #4,d3
0000024C        bne ausl1
00000250        addq.l #1,d4
00000252        cmpi.l #3,d4
00000258        bne ausl
0000025C        move.l a5,d5               ;Menü feststellen
0000025E        move.l #tab,a4
00000264        adda.l d5,a4
00000266        move.l (a4),a4             ;Menüpunkt feststellen
00000268        adda.l d6,a4
0000026A        move.l (a4),a4
0000026C        jmp (a4)                   ;und anspringen

0000026E tabmenu1   dc.l zaehler,nenner,parameter  ;Menüpunkte Menü1
0000027A tabmenu2   dc.l ausgabe,plotten,endof     ;Menüpunkte Menü2
00000286 tabmenu3   dc.l ZAEHLER,NENNER,FUNKTION   ;Menüpunkte Menü3

00000292 tab        dc.l tabmenu1,tabmenu2,tabmenu3 ;Menüs 1 bis 3
```

Jetzt haben wir das eigentliche Hauptprogramm schon hinter uns: Zunächst wird ein Screen geöffnet (Strukturen siehe unten), dessen Pointer in alle Strukturen der Windows eingetragen wird. Dann wird das Startwindow geöffnet und anschließend in dem Unterprogramm »PRINT« die Meldung in das Window hineingeschrieben. Nun wird das Menü aktiviert und alle Menüpunkte mit der universellen Routine aus Kapitel 2 wählbar gemacht. Beim Programmstart braucht diese Routine jedoch nicht durchlaufen zu werden, da durch die Funktion SetMenüStrip und unsere Menüstruktur automatisch alle Menüpunkte wählbar sind. Da der Punkt »Start« aber der Einsprung nach der Ausführung eines Menüpunktes ist, während der das Menü unwählbar gemacht wird, mußte die Routine an dieser Stelle eingesetzt werden.

Jetzt können wir auf eine Message warten und benutzen dazu den Weg mit der WAIT-Funktion. Der Task wird so lange »eingefroren«, bis der User irgendeinen Menüpunkt angewählt hat. Die Auswertung des Menüpunktes folgt, sobald ein Wert ungleich $FFFF vom Message-Port übergeben wurde. Dazu wird die Routine aus Kapitel 2 angewendet. In den Tabellen tabmenu1 bis tabmenu3 finden Sie die Einsprungadressen der einzelnen Menüpunkte.

Bevor diese angesprungen werden, machen wir noch alle Menüpunkte unwählbar. Dies ist notwendig, obwohl die Menüpunkte in der Regel ihr eigenes Window öffnen. Der Anwender kann nämlich durch geschicktes Verschieben des Startwindows dieses so legen, daß es nicht von dem Menüpunkt-Window überdeckt wird. Wenn er dann das Startwindow anklickt, wären die Menüpunkte für ihn theoretisch wieder wählbar, obwohl er sich gerade in einer Routine eines Menüpunktes befindet! Bei der Druckerausgabe, wo überhaupt kein anderes Fenster geöffnet wird, wäre dieses Problem natürlich noch viel eklatanter. Daher schalten wir es lieber ab, so daß alle Menüpunkte punktiert erscheinen. Die Labels in den Tabellen haben folgende Bedeutung:

tab:	tabmenu1	Adresse der Tabelle mit Adressen Menüpunkte Menü "Eingabe"
	tabmenu2	Adresse der Tabelle mit Adressen Menüpunkte Menü "Optionen"
	tabmenu3	Adresse der Tabelle mit Adressen Menüpunkte Menü "Grafik"
tabmenu1:	zaehler	Adresse Menüpunkt "Zähler eingeben"
	nenner	Adresse Menüpunkt "Nenner eingeben"
	parameter	Adresse Menüpunkt "Parameter eingeben"
tabmenu2:	ausgabe	Adr Menüpunkt "Ausgabe" mit Submenüs "Bildschirm, Drucker"
	plotten	Adresse Menüpunkt "Maßstab" mit Submenüs "1:1, Anpassung"
	endof	Adresse Menüpunkt "Ende"
tabmenu3:	Zaehler	Adresse Menüpunkt "Zähler plotten"
	Nenner	Adresse Menüpunkt "Nenner plotten"
	Funktion	Adresse Menüpunkt "Funktion plotten"

3.8.3 Menü »EINGABE«

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
          * ERSTES MENÜ (EINGABE)

         * Menüpunkt Zaehler eingeben

0000029E zaehler    move.b #32,d7        ;Strings für Eingabefelder
000002A2            lea fakzaehler,a3    ;Zähler vorbereiten
000002A8            lea fakx1,a4
000002AE            lea expzaehler,a5
000002B4            lea expx1,a6
000002BA zloop1     move.l (a3)+,(a4)+
000002BC            move.l (a5)+,(a6)+
000002BE            subq #1,d7
000002C0            bne zloop1
000002C4            lea zaehlwindow,a0   ;Window für Zählereingabe
000002CA            bsr  mainwindow      ;öffnen
000002CE            move.b #32,d7
000002D2            lea fakzaehler,a3
000002D8            lea fakx1,a4         ;Eingabestrings sichern
000002DE            lea expzaehler,a5
000002E4            lea expx1,a6
000002EA zloop2     move.l (a4)+,(a3)+
000002EC            move.l (a6)+,(a5)+
000002EE            subq #1,d7
000002F0            bne zloop2
000002F4            lea floatfakz,a3     ;Faktoren in Fließkomma
000002FA            bsr  wandelfak       ;umwandeln
000002FE            lea floatexpz,a3     ;Exponenten in Fließkomma
00000304            bsr  wandelexp       ;umwandeln
00000308            bra  exit            ;fertig

         * Menüpunkt Nenner eingeben

0000030C nenner     move.b #32,d7        ;s.o.
00000310            lea faknenner,a3
00000316            lea fakx1,a4
0000031C            lea expnenner,a5
00000322            lea expx1,a6
00000328 nloop1     move.l (a3)+,(a4)+
0000032A            move.l (a5)+,(a6)+
0000032C            subq #1,d7
0000032E            bne nloop1
00000332            lea nennwindow,a0
00000338            bsr  mainwindow
```

```
0000033C           move.b #32,d7
00000340           lea faknenner,a3
00000346           lea fakx1,a4
0000034C           lea expnenner,a5
00000352           lea expx1,a6
00000358 nloop2    move.l (a4)+,(a3)+
0000035A           move.l (a6)+,(a5)+
0000035C           subq #1,d7
0000035E           bne nloop2
00000362           lea floatfakn,a3
00000368           bsr  wandelfak
0000036C           lea floatexpn,a3
00000372           bsr  wandelexp
00000376           bra  exit
         * Menüpunkt Parameter eingeben

0000037A parameter move.b #12,d7         ;Strings für Eingabe
0000037E           lea Unterex1,a3       ;vorbereiten
00000384           lea Untere,a4
0000038A ploop1    move.l (a3)+,(a4)+
0000038C           subq #1,d7
0000038E           bne ploop1
00000392           lea parawindow,a0     ;Window für Eingabe öffnen
00000398           bsr  mainwindow
0000039C           move.b #12,d7         ;Eingabestrings sichern
000003A0           lea Unterex1,a3
000003A6           lea Untere,a4
000003AC ploop2    move.l (a4)+,(a3)+
000003AE           subq #1,d7
000003B0           bne ploop2
000003B4           lea floatuntere,a3    ;und in Fließkomma
000003BA           bsr  wandelpar        ;umwandeln
000003BE           lea propinfo,a0       ;Proportionalwert
000003C4           clr.l d0              ;holen
000003C6           move.w 2(a0),d0
000003CA           cmpi.l #$C000,d0
000003D0           bls notdrei
000003D4           move.b #3,stellen     ;3 Nachkommastellen
000003DC           bra  parend           ;bei 75-100 %
000003E0 notdrei   cmpi.l #$4000,d0      ;2 Nachkommastellen
000003E6           bls notzwei           ;bei 25-75 %
000003EA           move.b #2,stellen
000003F2           bra  parend           ;1 Nachkommastelle
000003F6 notzwei   move.b #1,stellen     ;bei 0-25 %
000003FE parend    bra  exit

00000402 stellen   dc.b 1
                   even
```

Das erste Menü enthält wie schon erwähnt die Eingaberoutinen für Faktoren und Exponenten von Zähler sowie Nenner und die Parameter. Zunächst zu den ersten beiden Menüpunkten. Normalerweise würde man so vorgehen, daß man jeweils für Zähler und Nenner getrennte Gadget-Strukturen verwendet. Insgesamt wären jeweils acht Gadgets für die Eingabe der Faktoren und acht für die der Exponenten notwendig, insgesamt also jeweils 16 Stringgadgets. Da jede Gadget-Struktur 44 Byte und jede dazu erforderliche String-Info-Struktur 36 Byte enthält, wären für Zähler und Nenner zweimal 16*(44+36) = 1280 Byte erforderlich. Zusätzlich müssen wir noch 2*16 Stringpuffer für 2560 Eingabetexte zur Verfügung stellen. Dafür wollen wir je 16 Zeichen reservieren, so daß nochmals 2*16*16 = 512 Byte erforderlich sind, summa summarum daher 3072 Byte. Diese Zahl sieht zwar auf den ersten Blick gewaltig aus, unser Programmieraufwand ist jedoch durch die Verwendung von Makros minimal geblieben.

Trotzdem ist es eine Überlegung wert, ob man nicht doch mit weniger Quelltext und Speicherplatz auskommt, zumal der Aufbau der Zähler- und Nennereingaben völlig identisch ist. Ein Trugschluß wäre es zu glauben, man könne ohne weiteres nur mit 16 Eingabegadets auskommen, die von beiden Eingaberoutinen benutzt werden könnten, da sich die Eingabepuffer wechselweise mit Daten überschreiben würden. Trotzdem wäre es schön, dieses realisieren zu können, da man nur noch 1536 Byte benötigen würde. Die Lösung besteht darin, tatsächlich nur einmal 16 Stringgadgets zu implementieren, dafür aber drei Eingabepuffer zu installieren. Damit kann man wie folgt vorgehen: Die Gadget-Struktur erhält wie normal einen Eingabepuffer mit 16 * 16 Zeichen Platz gestellt. Zusätzlich erhalten aber Zähler und Nenner jeweils ihren eigenen Pseudo-Puffer der gleichen Größe, in den natürlich keine Eingaben erfolgen können, da sie von keinen Gadgets bedient werden. Wenn man aber vor dem Aufruf der Stringgadets den Inhalt des eigenen Puffers in den Gadget-Puffer kopiert und dessen Inhalt nach den Eingaben wieder zurückkopiert, kann man mit einer Gadget-Struktur auskommen. Man benötigt dann nur noch einmal 16*(44+36) = 1280 Byte für die Gadget-Strukturen sowie dreimal 16*16 = 768 Byte für die drei Eingabepuffer, zusammen also nur noch 2048 Byte. Damit wurden gegenüber der ersten Version mit getrennten Gadgets für Zähler und Nenner genau 1024 Byte eingespart. Der Quelltextaufwand bleibt dabei ungefähr konstant, da einerseits 32 Strukturen weniger angelegt, andererseits aber auch die Routinen für die Kopieroperationen in den Eingabepuffern programmiert werden müssen. In unserem Programm sind die Startadressen der einzelnen Puffer so bezeichnet:

fakx1	Eingabepuffer der acht Faktoren
expx1	Eingabepuffer der acht Exponenten
fakzaehler	Puffer für die acht Faktoren des Zählers
expzaehler	Puffer für die acht Exponenten des Zählers
faknenner	Puffer für die acht Faktoren des Nenners
expnenner	Puffer für die acht Exponenten des Nenners

Nach dieser Einführung wird es für Sie ein leichtes sein, liebe Leser, die Routinen der einzelnen Menüpunkte nachzuvollziehen: Als erstes haben wir die Zählereingabe (Label »zaehler«). Hier werden zunächst die Pufferinhalte in den Eingabepuffer für die Gadgets

kopiert. Dann wird das Window geöffnet. Die Routine »mainwindow«, die wir unten noch genauer beschreiben, hat die Aufgabe, einerseits das Fenster zu öffnen und andererseits solange zu warten, bis entweder das Close- oder das Ende- Gadget angeklickt wurden. Nun wird der Eingabepuffer wieder zurückkopiert. Als letztes werden schließlich die Eingabestrings ins Fließkommaformat umgewandelt. Dazu dienen die Routinen »wandlfak«, die die Faktoren umwandelt, und »wandelexp«, die gleiches mit den Exponenten vollzieht. Mit den Namen »floatfakz« und »floatexpz« sind die Anfänge der Fließkommapuffer bezeichnet, in denen die FFP-Zahlen abgelegt werden.

Damit sind wir auch schon am Ende des Zählereingabeteils angelangt. Der entsprechende Teil für den Nenner funktioniert analog. Die Unterschiede beschränken sich auf die Label, »faknenner« und »expnenner« für die Stringpuffer und »floatfakn« sowie »floatexpn« für die Fließkommapuffer.

Nun kommen wir zu den Parametern. Die Festlegung der unteren und oberen Grenze sowie der Schrittweite erfolgt ebenfalls in Stringgadgets. Da diese nur einmal belegt sind, werden Sie sich wundern, daß auch hier zwei Eingabepuffer für jedes Gadget vorhanden sind, obwohl eins doch ausreichen müßte. Dies tut es auch, aber erinnern wir uns an unsere Umwandlungsroutine von String nach Fließkomma: Diese verändert den String zur Auswertung, indem Vorzeichen und Dezimalpunkt eleminiert werden. Dies hätte bei nur einem Puffer zur Folge, daß eine Eingabe der Form

"-2.56"

nach der Umwandlung als

"256"

in dem Puffer stände, was an sich ja nicht schlimm wäre, da uns nach der Umwandlung ja nicht mehr der String, sondern die FFP-Zahl interessiert. Das Problem besteht aber darin, daß bei einer weiteren Eingabe diese »256« auf dem Bildschirm erschiene, da ja immer der aktuelle Inhalt des Eingabepuffers angezeigt wird! Somit müssen wir doch wieder den alten Inhalt des Puffers direkt nach der Eingabe sichern, bevor dessen Inhalt verstümmelt werden kann. Daher wird Ihnen der Teil von dem Label »parameter« an bekannt vorkommen. Neu ist hingegen die Auswertung des Proportionalgadgets. Wie Sie aus Kapitel 2.4. wissen, schreibt Intuition in das Proportional-Info ab der Adresse 2 einen Wert zwischen 0 und $FFFF je nach Stellung des Reglers. Da wir nur drei Auswahlmöglichkeiten für die Anzahl der Nachkommastellen haben, müssen wir diesen Bereich sinnvoll aufteilen: Ich halte es für angebracht, wenn man den Werten »1« und »3« jeweils 25 % des Reglerbereiches von links bzw. rechts zuordnet, während der mittlere Teil von 25 % bis 75 % dem Wert »2« entsprechen soll: Daher wird in unserem Programm eine Abfrage bezüglich der Werte $4000 (=25 %) und $C000 (=75 %) durchgeführt und die entsprechende Anzahl der Nachkommastellen in dem Label »stellen« abgespeichert. Dies geht so vor sich: Wenn der Proportional-Wert größer als $C000 ist, wird sofort auf drei Stellen erkannt. Als nächstes wird geprüft, ob der Wert kleiner als $4000 ist, was bedeuten würde, daß nur eine Stelle gewünscht wird. Es ist ersichtlich, daß nicht nur die Eingabe, sondern auch die Auswertung

eines Proportionalgadgets sehr einfach vor sich gehen kann. Als Startwerte für die erste Eingabe wurden folgenden Werte festgelegt:

Faktoren Zähler	: jeweils 1
Faktoren Nenner	: jeweils 1
Exponenten Zähler	: jeweils von 0 bis 7 aufsteigend
Exponenten Nenner	: jeweils von 0 bis 7 aufsteigend
Untere Grenze	: −2
obere Grenze	: +2
Schrittweite	: 0.5
Zahl Nachkommastellen	: 1

Unten werden wir die einzelnen Strukturen finden. Wenn Ihnen diese Anfangswerte nicht gefallen, können Sie sie natürlich beliebig abändern.

3.8.4 Funktionswertausgabe auf Bildschirm

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * ZWEITES MENÜ (OPTION)

         * Menüpunkt Ausgabe

00000404 ausgabe     move.l #subtab,a4          ;Untermenü holen
0000040A             adda.l d7,a4
0000040C             move.l (a4),a4
0000040E             jmp (a4)

00000410 subtab      dc.l  schirm,printer

         * Untermenüpunkt Bildschirmausgabe

00000418 schirm      bsr  berechnung            ;Polynom berechnen
0000041C             lea auswindow,a0           ;Window für Ausgabe öffnen
00000422             CALLINT OpenWindow
0000042C             move.l d0,iwindowpointer
00000432             clr.w startpos             ;Nummer erstes Wertepaar
00000438 schleife1   move.l iwindowpointer,a0
0000043E             move.l 50(a0),a1           ;Rastport holen
00000442             clr.l d0                   ;Modus löschen:
00000444             CALLGRAF SetAPen           ;Zeichen=Hintergrundfarbe
0000044E             move.l #2,d0               ;Ausgabebereich löschen:
00000454             move.l #15,d1
0000045A             move.l #315,d2             ;von (x,y)=(2,15)
00000460             move.l #130,d3             ;nach (315,130)
00000466             CALLGRAF RectFill
```

```
00000470            move.w startpos,position ;Positiom in Wertetab.
0000047A            move.b #20,zeilen        ;y-Startkoordinate
00000482            clr.b zähler             ;Zähler Wertepaare
00000488 schleife   clr.l d0
0000048A            move.w position,d0       ;Position*4
00000490            lsl.w #2,d0              ;(Fließkomma=Langwort)
00000492            lea xwert,a0             ;Start Wertetabelle x
00000498            move.l (a0,d0),d3        ;plus Position=Wert holen
0000049C            lea ywert,a0             ;das gleiche mit y
000004A2            move.l (a0,d0),d4
000004A6            move.b zeilen,zeile      ;aktuelle y-Koordinate
000004B0            bsr  xout                ;x-Wert ausgeben
000004B4            move.l d4,d3
000004B6            bsr  yout                ;y-Wert ausgeben
000004BA            addi.b #1,zähler         ;Schon 9 Wertepaare?
000004C2            cmpi.b #9,zähler         ;(Dann Bildschirm voll)
000004CA            bhi schlende             ;Ja, Ausgabe Stop
000004CE            addq.w #1,position
000004D4            move.w position,d0       ;Alle Wertepaare ausgeben?
000004DA            move.w paarzahl,d1
000004E0            cmp.w d0,d1
000004E2            beq schlende             ;ja, Ausgabe Stop
000004E6            addi.b #10,zeilen        ;y-Koordinate erhöhen
000004EE            bra  schleife            ;nächste Werte ausgeben

000004F2 schlende   bsr  iloop               ;auf Message warten

000004F6            cmpi.w #113,d2           ;Scroll abwärts?
000004FA            bne sch1
000004FE            move.w startpos,d3       ;ja, aber wenn alle Werte
00000504            addi.w #10,d3            ;ausgegeben, kein Scrollen
00000508            move.w paarzahl,d4       ;mehr möglich
0000050E            cmp.w d4,d3
00000510            bge schlende
00000514            addi.w #10,startpos
0000051C            bra  schleife1
00000520 sch1       cmpi.w #112,d2           ;Scrollen aufwärts?
00000524            bne sch2
00000528            move.w startpos,d3       ;ja, aber wenn erste Werte
0000052E            beq schlende             ;auf Bildschirm, kein
00000532            subi.w #10,startpos      ;Scrollen mehr möglich
0000053A            bra  schleife1
0000053E sch2       bra  exit                ;Ausgabe beenden

00000542 position   dc.w 0
00000544 startpos   dc.w 0
00000546 zeilen     dc.b 0
```

```
                        even
00000548 zähler         dc.b 0
                        even
```

Das zweite Menü startet mit dem Menüpunkt »Ausgabe«. Da dieser zwei Untermenüpunkte enthält, müssen diese zunächst ausgewertet werden, was genauso geschieht, wie die Auswertung der Menüs und Menüpunkte (siehe oben)

Der Menüpunkt »schirm« dient dazu, die berechneten Werte auf dem Bildschirm auszugeben. Damit bei einer Anzahl von Werten, die die Bildschirmkapazität überschreiten würde, nicht einfach weiterausgegeben wird, wenn der Schirm gefüllt ist, werden die Werte »häppchenweise« ausgegeben. Durch Anwählen zweier Gadgets kann man dann weitere oder aber auch vorhergehende Werte ausgeben. Nachdem die y-Werte in dem Unterprogramm »berechnung« in ein Array geschrieben wurden, wird das Ausgabefenster geöffnet. Dann wird der Teil des Windows für die Ausgabe gelöscht. Da Intuition leider keinen Befehl bereithält, mit dem man ein Window nur teilweise löschen kann, mußte zu einem Trick gegriffen werden: Nachdem die Zeichenfarbe mit der SetAPen-Funktion auf die Hintergrundfarbe eingestellt wurde, wurde das gewünschte Rechteck mit der RectFill-Funktion ausgefüllt gezeichnet. Weil dieses mit der Hintergrundfarbe geschah, kann man auch davon sprechen, daß das Rechteck gelöscht wurde. Nun kann die Ausgabe beginnen. Zunächst wurden drei Variablen mit folgender Bedeutung deklariert:

zeilen: Enthält die aktuelle Ausgabezeile (Startwert: 20)
zähler: Zählt die auf dem Bildschirm ausgegebenen Wertepaare
position: Zeiger auf Tabelle mit FFP-Werten

Jetzt können die ersten neun Wertepaare in den Routinen »xout« und »yout« auf den Bildschirm gebracht werden. Es wird anschließend so lange gewartet, bis der Anwender entweder eines der Scroll-Gadgets, das Close-Gadget oder das Ende-Gadget angewählt hat. Wurde ein Scroll-Gadget angeklickt, werden die nächsten bzw. vorhergehenden zehn Wertepaare ausgegeben. Falls ein Scrolling nicht mehr möglich ist, weil entweder der erste oder letzte Wert des Arrays erreicht wurde, bleibt das Anklicken ohne Wirkung.

Dieses wird durch folgende Programmierung erreicht: Die Abfrage, ob der Bildschirm gefüllt ist, geschieht mit der Variablen »zaehler«, die bei jedem ausgegebenen Wertepaar um eins erhöht wird. Bei jedem neuen Bildschirmaufbau wird sie daher auf Null gesetzt. Die Variable »position« gibt an, wo sich im Array die auszugebenden Werte befinden. Dafür wird sie mal vier genommen (eine FFP-Zahl besteht ja aus einem Langwort) und ebenfalls bei jeder Ausgabe um eins erhöht. Natürlich läuft sie chronologisch weiter und wird nicht neu bei jedem Bildschirmaufbau initialisiert. Die Variable »startpos« gibt schließlich an, welcher Wert am Anfang jedes Bildschirms steht. Sie wird beim Scrollen immer in Zehnerschritten erhöht oder erniedrigt und gibt praktisch den Wert an, von dem an die folgenden Werte ausgegeben werden, wobei wieder die Variable »position« in Aktion tritt. Es ist erstaunlich, was für eine komfortable Ausgabe man mit 50 Assemblerzeilen programmieren kann, nicht wahr?

3.8.5 Funktionswertausgabe auf Drucker

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
        * Untermenüpunkt Druckerausgabe

0000054A printer    bsr  berechnung        ;Werte berechnen
0000054E            move.l #printername,d1
00000554            move.l #1006,d2        ;Druckerkanal öffnen
0000055A            CALLDOS Open
00000564            tst.l d0
00000566            beq perror             ;Fehler aufgetreten
0000056A            move.l d0,printerpointer
00000570            bsr  über              ;Überschrift ausgeben
00000574            clr.w position
0000057A pschleife  clr.l d0
0000057C            move.w position,d0     ;Werte aus Tabelle holen
00000582            lsl.w #2,d0
00000584            lea xwert,a0
0000058A            move.l (a0,d0),d3
0000058E            lea ywert,a0           ;(s.o.)
00000594            move.l (a0,d0),d4
00000598            bsr  xdruck            ;x für Druck aufbereiten
0000059C    .       move.l d4,d3
0000059E            bsr  ydruck            ;y für Druck aufbereiten
000005A2            bsr  druck             ;Wertepaar ausdrucken
000005A6            addq.w #1,position
000005AC            move.w position,d1     ;Alle Werte ausgedruckt?
000005B2            move.w paarzahl,d2
000005B8            cmp.w d2,d1            ;nein, weiterdrucken
000005BA            blt pschleife

000005BE            move.l printerpointer,d1;Druckerkanal schließen
000005C4            CALLDOS Close
000005CE            bra  exit1             ;fertig

        * Fehlerbehandlung

000005D2 perror     bsr  requester         ;Requester ausgeben
000005D6            cmpi.l #1,d0           ;Nochmal versuchen?
000005DC            beq printer            ;Ja, bitte schön!
000005E0            bra  exit1             ;nein, Drucker kaputt

000005E4 printername     dc.b "PRT:",0
                         even

000005EA printerpointer  dc.l 0
                         even
```

Die Druckerausgabe ist einfacher zu realisieren, da man dort alle Werte der Reihe nach ausgeben kann ohne auf einen etwaigen gefüllten Bildschirm achten zu müssen. Zunächst wird dazu ein DOS-Kanal (siehe Kapitel 5) geöffnet, besser gesagt, es wird versucht, einen Kanal zu öffnen. Wenn dieses fehlschlägt, gibt die OPEN-Funktion den Wert Null zurück, worauf zum Label »perror« verzweigt wird. Hier wird ein Requester aufgerufen, das dem Anwender zwei Möglichkeiten zur Verfügung stellt: Entweder kann ein erneuter Versuch unternommen oder abgebrochen werden. Falls jedoch ein Kanal korrekt geöffnet wurde, wird zunächst die Überschrift der Tabelle in dem Unterprogramm »ueber« ausgegeben. Dann werden nach und nach alle Wertepaare aus dem Array geholt und ausgegeben, wie es prinzipiell auch schon auf dem Bildschirm geschah. Schließlich wird der Druckerkanal wieder geschlossen, womit der Menüpunkt »Ausgabe« abgehakt werden kann.

3.8.6 Maßstab für Grafikausgabe festlegen

```
         * Menüpunkt Maßstab
000005EE plotten         move.l #subtabl,a4      ;Untermenü
000005F4                 adda.l d7,a4
000005F6                 move.l (a4),a4
000005F8                 jmp (a4)

000005FA subtabl         dc.l  M1zu1,anpassung

         * Untermenüpunkt Maßstab 1:1

00000602 M1zu1           move.b #1,masstab       ;Flag setzen
0000060A                 bra  exit1

         * Untermenüpunkt Maßstab Anpassung

0000060E anpassung       clr.b masstab           ;Flag setzen
00000614                 bra  exit1

00000618 masstab         dc.b 1
                         even

         * Menüpunkt Ende

0000061A endof           bra  ende
```

Im Menüpunkt »Maßstab« wird schlicht und einfach der Ausgabemaßstab für die Grafik in der Variablen »masstab« festgelegt, so daß weiteres genausowenig gesagt werden muß wie zum Menüpunkt »Ende«, mit dem das Programm verlassen wird.

3.8.7 Das Grafik-Menü

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * DRITTES MENÜ (PLOTTEN)

         * Menüpunkt Zähler

0000061E ZAEHLER         bsr  plotscreen         ;Plotscreen öffnen
00000622                 move.l #zaehlwert,d7
00000628                 bsr  xmasstab           ;Maßstab festlegen
0000062C                 lea zaehlwert,a3
00000632                 bsr  zeichnen           ;Plotten
00000636                 bsr  iloop              ;auf Message warten
0000063A                 bra  exit2

         * Menüpunkt Nenner

0000063E NENNER          bsr  plotscreen         ;s.o.
00000642                 move.l #nennwert,d7
00000648                 bsr  xmasstab
0000064C                 lea nennwert,a3
00000652                 bsr  zeichnen
00000656                 bsr  iloop
0000065A                 bra  exit2

         * Menüpunkt Funktion

0000065E FUNKTION        bsr  plotscreen         ;s.o.
00000662                 move.l #ywert,d7
00000668                 bsr  xmasstab
0000066C                 lea ywert,a3
00000672                 bsr  zeichnen
00000676                 bsr  iloop
0000067A                 bra  exit2

         * Plotscreen öffnen

0000067E plotscreen      lea plot_defs,a0        ;Screen öffnen
00000684                 CALLINT OpenScreen
0000068E                 move.l d0,plotpointer   ;Pointer sichern
00000694                 move.l d0,point6
0000069A                 lea plotwindow,a0       ;Window öffnen
000006A0                 CALLINT OpenWindow
000006AA                 move.l d0,iwindowpointer ;Pointer sichern
000006B0                 rts
```

Da unser Programm modular aufgebaut ist, sind die eigentlichen Hauptroutinen wieder sehr kurz, die eigentliche Arbeit wird in Unterprogrammen verrichtet. Zunächst müssen Sie wissen, liebe Leser, daß die Routine »berechnung« nicht nur x- und y-Werte in einem Array ablegt, sondern auch die Ergebnisse des Zähler- sowie Nennerpolynoms dort abspeichert. Daher können wir bei der Grafikausgabe leicht ohne weitere Rechnungen auf diese Werte zugreifen. Zunächst wird von den identisch aufgebauten Routinen der Menüpunkte ein Screen in der x-Auflösung von 640 Punkten geöffnet. Dies geschieht genauso in der Routine »plotscreen« wie das anschließende Öffnen eines Windows für die grafische Ausgabe. Dann wird in der Routine »xmasstab« die Schrittweite zwischen je zwei x-Punkten berechnet, die sich danach richtet, was der User als untere und obere Grenze eingegeben hat. Während die Ausgabe in dem Unterprogramm »zeichnen« erfolgt, kennen Sie bereits die Subroutine illop: Sie holt eine Message und wertet sie aus. In dem Grafikteil ist es nur interessant, ob das Close-Gadget angeklickt wurde, worauf der Menüpunkt verlassen wird. Statt wie üblich nach »exit1« zu springen, muß hier noch das Window und der Screen geschlossen werden, was ab »exit2« vollzogen wird.

3.8.8 Maßstäbe für Grafikausgabe berechnen

```
         * Schrittweite für 1 Pixel festlegen
000006B2 xmasstab    move.l floatobere,d0        ;obere Grenze
000006B8             move.l floatuntere,d1       ;minus
000006BE             CALLFFP SPSub
000006C8             move.l d0,d2                ;untere Grenze
000006CA             move.l #640,d0
000006D0             CALLFFP SPFlt
000006DA             move.l d0,d1
000006DC             move.l d0,xmax
000006E2             move.l d2,d0
000006E4             CALLFFP SPDiv               ;Ergebnis/640
000006EE             move.l floatschritt,sicher  ;Schrittweite retten
000006F8             move.l d0,floatschritt
000006FE             move.l d0,floatxschritt
00000704             move.l d0,floatyschritt
0000070A             move.l d7,sicher1
00000710             bsr  berechnung             ;Daten berechnen
00000714             move.l sicher1,d7
0000071A             move.l #10,d0
00000720             CALLFFP SPFlt               ;ymin = 10
0000072A             move.l d0,ymin
00000730             move.l #256,d0
00000736             CALLFFP SPFlt               ;ymax = 256
00000740             move.l d0,ymax
00000746             move.l #133,d0
0000074C             CALLFFP SPFlt               ;Nullinie (x-Achse):
00000756             move.l d0,null
```

```
0000075C            move.b masstab,d0          ;y = 133
00000762            cmpi.b #1,d0               ;1:1, dann fertig
00000766            beq massfertig
0000076A            bsr  minimax               ;Anpassung
0000076E massfertig rts

00000770 sicher     dc.l 0
00000774 sicher1    dc.l 0
00000778 xmax       dc.l 0
0000077C ymin       dc.l 0
00000780 ymax       dc.l 0
00000784 null       dc.l 0

         * Maximum und Minimum feststellen, daraus Maßstab berechnen

00000788 minimax         move.l d7,a5
0000078A                 move.w #639,d3        ;Zähler
0000078E                 move.l (a5),max
00000794                 move.l (a5)+,min      ;Anfangswerte als
0000079A minimaxloop     move.l (a5)+,d1       ;Minimum und
0000079C                 move.l d1,d2          ;Maximum setzen
0000079E                 move.l max,d0         ;nächsten Wert
000007A4                 CALLFFP SPCmp         ;vergleichen
000007AE                 cmpi.l #-1,d0         ;kleiner gleich,
000007B4                 beq nomax             ;kein neues Maximum
000007B8                 move.l d2,max         ;neues Maximum
000007BE                 bra  minimaxend
000007C2 nomax           move.l d2,d1          ;wieder vergleichen
000007C4                 move.l min,d0         ;größer gleich,
000007CA                 CALLFFP SPCmp         ;kein neues Minimum
000007D4                 cmpi.l #1,d0
000007DA                 beq minimaxend
000007DE                 move.l d2,min         ;neues Minimum
000007E4 minimaxend      subq.w #1,d3          ;nächsten Wert
000007E6                 bne minimaxloop       ;untersuchen
000007EA                 move.l max,d0
000007F0                 move.l min,d1         ;Schrittweite=
000007F6                 CALLFFP SPSub
00000800                 move.l d0,d7          ;(Max-Min)\
00000802                 move.l ymax,d0        ;(ymax-ymin)
00000808                 move.l ymin,d1
0000080E                 CALLFFP SPSub
00000818                 move.l d0,d1
0000081A                 move.l d7,d0
0000081C                 CALLFFP SPDiv
00000826                 move.l d0,floatyschritt
0000082C                 rts
```

```
0000082E min            dc.l 0
00000832 max            dc.l 0
```

In der Routine »xmasstab« wird die Schrittweite für den Abstand zwischen je zwei x-Punkten festgelegt. Hierfür gilt:

xschritt = (obere Grenze – Untere Grenze)/640

Anschließend werden die Wertepaare berechnet und einige wichtige Konstanten definiert:

ymin, ymax : 10,256 maximal zulässige y-Koordinaten
null : 133, Lage der x-Achse (mittig im Ausgabefenster)

Falls im Menü »Maßstab« dieser zu 1:1 definiert wurde, sind wir schon fertig, da der x- auch automatisch dem y-Maßstab entspricht und damit auch eine gleiche Schrittweite existiert. Falls man aber »Anpassung« gewählt hat, muß der y-Maßstab so gewählt werden, daß alle y-Punkte auf dem Bildschirm dargestellt werden können. Dazu wird die Routine »minimax« aufgerufen. Die y-Schrittweite ergibt sich nun zu

yschritt = (maximaler y-Wert – minimaler y-Wert)/(ymax – ymin)

Zunächst müssen also Minimum und Maximum aller berechneten y-Werte herausgefunden werden. Dazu wird der erste Wert im Array sowohl als Minimum als auch als Maximum definiert. Dann werden nacheinander alle 639 folgenden Werte jeweils mit Minimum und Maximum verglichen. Falls sie größer bzw. kleiner sind, nehmen sie den Platz ihres Vorgängers ein. Somit hat man zum Schluß garantiert den größten und kleinsten Wert herausgefiltert. Schließlich wird noch die Division durch die Differenz der maximal und minimal zulässigen y-Koordinaten durchgeführt und die neue Schrittweite festgelegt.

3.8.9 Funktionswerte plotten

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         *Daten in Window plotten
00000836 zeichnen    move.l iwindowpointer,a0
0000083C             move.l 50(a0),d6          ;Rastport holen
00000840             move.b masstab,d0
00000846             cmpi.b #1,d0              ;Maßstab 1:1?
0000084A             beq noverzerr             ;ja, ok
0000084E             move.l min,d0             ;Bei Anpassung:
00000854             move.l floatyschritt,d1   ;y-Nullinie=
0000085A             CALLFFP SPDiv
00000864             move.l ymax,d1            ;(Min\Schrittweite)
0000086A             CALLFFP SPAdd             ;+ymax
00000874             move.l d0,null
```

```
0000087A noverzerr   move.l null,d0          ;Keine x-Achse
00000880             CALLFFP SPFix           ;zeichen, wenn:
0000088A             cmpi.l #256,d0
00000890             bge noxlinie            ;y größer 256
00000894             cmpi.l #10,d0
0000089A             bls noxlinie            ;oder y kleiner 10
0000089E             move.l d0,d4
000008A0             move.l d6,a1
000008A2             clr.l d0
000008A4             move.l d4,d1            ;x-Achse zeichnen
000008A6             CALLGRAF Move           ;von (0,200) nach
000008B0             move.l d6,a1            ;(639,200)
000008B2             move.l #639,d0
000008B8             move.l d4,d1
000008BA             CALLGRAF Draw
000008C4 noxlinie    move.l floatuntere,d0   ;Startzahl negieren
000008CA             CALLFFP SPNeg
000008D4             move.l floatxschritt,d1 ;durch Maßstab
000008DA             CALLFFP SPDiv
000008E4             move.l d0,d2            ;=x-Offset
000008E6             clr.l d1
000008E8             CALLFFP SPCmp           ;für y-Achse:
000008F2             cmpi.l #1,d0
000008F8             beq noylinie            ;nicht zeichnen,
000008FC             move.l d2,d0
000008FE             move.l xmax,d1          ;wenn x kleiner 0
00000904             CALLFFP SPCmp
0000090E             cmpi.l #-1,d0           ;oder x größer 639
00000914             beq noylinie
00000918             move.l d2,d0
0000091A             CALLFFP SPFix           ;in Integer wandeln
00000924             move.l d0,d2
00000926             move.l ymax,d0
0000092C             CALLFFP SPFix
00000936             move.l d6,a1
00000938             move.l d0,d1            ;y-Achse zeichnen
0000093A             move.l d2,d0
0000093C             CALLGRAF Move
00000946             move.l ymin,d0          ;von (x,390) nach
0000094C             CALLFFP SPFix
00000956             move.l d0,d1
00000958             move.l d6,a1            ;(x,10)
0000095A             move.l d2,d0
0000095C             CALLGRAF Draw

00000966 noylinie    clr.l d2                ;x-Koordinate
00000968             move.l #640,d3          ;Zaehler
```

```
0000096E               clr.l d7
00000970               move.b masstab,d5
00000976 plotloop      move.l (a3)+,d0          ;Wert holen
00000978               move.l floatyschritt,d1  ;Maßtab
0000097E               CALLFFP SPDiv            ;dividieren
00000988               tst.b d5                 ;Anpassung?
0000098A               beq nofaktor             ;nicht quetschen!
0000098E               move.l #$A0000040,d1
00000994               CALLFFP SPMul
0000099E nofaktor      move.l d0,d1
000009A0               move.l null,d0           ;Nullinie
000009A6               CALLFFP SPSub
000009B0               move.l d0,d4             ;hinzuaddieren=Koordinate
000009B2               move.l ymax,d1           ;Maximum
000009B8               CALLFFP SPCmp            ;vergleichen
000009C2               cmpi.l #-1,d0            ;größer?
000009C8               bne plot1                ;ja, kein Punkt
000009CC               move.l #255,d0
000009D2               bra  plot2
000009D6 plot1         move.l d4,d0
000009D8               move.l ymin,d1          ;Minimum
000009DE               CALLFFP SPCmp           ;vergleichen
000009E8               cmpi.l #1,d0            ;kleiner
000009EE               bne okplot              ;ja, kein Punkt
000009F2               move.l #10,d0
000009F8 plot2         move.l d7,d4
000009FA               clr.l d7
000009FC               bra  plot3
00000A00 okplot        move.l d4,d0
00000A02               CALLFFP SPFix
00000A0C               move.l d7,d4
00000A0E               move.l #1,d7
00000A14 plot3         move.l d6,a1            ;Rastport holen
00000A16               move.l d0,d1            ;x-Koordinate
00000A18               move.l d2,d0            ;y-Koordinate
00000A1A               tst.l d4
00000A1C               beq punkt               ;Linie vom letzten
00000A20               CALLGRAF Draw           ;Punkt zeichnen
00000A2A               bra  noplot1
00000A2E punkt         CALLGRAF Move           ;Punkt zeichnen
00000A38 noplot1       addq #1,d2
00000A3A               subq #1,d3
00000A3C               bne plotloop
00000A40               move.l sicher,floatschritt
00000A4A               rts

00000A4C floatxschritt   dc.l 0
00000A50 floatyschritt   dc.l 0
```

Das Unterprogramm zum Zeichnen der Funktion arbeitet nach folgendem Schema: Zunächst wird die Lage der Nullinie festgelegt. Wenn der 1:1 Maßstab aktiviert ist, bleibt die Bildschirmmitte bestehen. Falls jedoch Anpassung vorliegt, errechnet sich die y-Koordinate zu

ynull = ymax – (ymin/yschritt)

mit	ynull	: y-Koordinate der Nullinie (0–256)
	ymax,ymin	: Maximal/minimal zulässige y-Koordinaten (256,0)
	yschritt	: y-Schrittweite im FFP-Format

Bei der Schrittweite handelt es sich um denjenigen Wert, der durch einen Sprung um einen Grafikpunkt in y-Richtung repräsentiert wird. Daraufhin wird überprüft, ob sich der errechnete Wert im zulässigen Bereich von ymin bis ymax bewegt. Falls dies zutrifft, wird die x-Achse gezeichnet. Als nächstes wird die y-Achse gezeichnet, falls sich der Nullpunkt innerhalb des betrachteten x-Bereiches befindet. Dazu wird überprüft, ob sich der Ausdruck

(– untere Grenze)/xschritt

in dem Bereich von 0 bis 639 bewegt. Falls die untere Grenze positiv ist, wird der Ausdruck negativ, die y-Achse würde nicht gezeichnet. Dies leuchtet auch sofort ein. Wenn die y-Achse dann gezeichnet wurde, beginnt in einer Schleife die Ausgabe der Grafikpunkte. Die y-Koordinaten berechnen sich dabei durch folgende Formel:

y = ynull + (ywert/yschritt) mit ywert: Funktionswert aus Array

Eine Sache ist dabei zu beachten: Falls der Maßstab 1:1 gewählt wurde, möchte der Anwender ein originalgetreues Abbild der Funktion auf dem Bildschirm sehen. Durch die unterschiedlichen Auflösungen in x- und y-Richtung sowie das rechteckige Bildschirmformat würde eine mathematisch korrekt berechnete Funktion verzerrt dargestellt werden. Deshalb wird der y-Wert mit dem Korrekturfaktor 0.625 multipliziert, bevor er ausgegeben wird. Der erforderliche Faktor kann aber von Bildschirm zu Bildschirm differieren, Sie können ihn nach Belieben ändern, aber bitte immer im FFP-Format!

Als letzte Operation vor der Ausgabe wird geprüft, ob der Punkt innerhalb der zulässigen Grenzen von ymin bis ymax liegt. Im Fall der Anpassung ist dies natürlich immer der Fall, da der Maßstab nach den Extremen der Funktionswerte ausgerichtet wurde. Zum Schluß werden die Daten vom FFP- ins Integer-Format umgewandelt und ausgegeben. Dabei wird der erste Punkt gesetzt, alle anderen Punkte werden durch Linien mit dem vorhergehenden verbunden. Hierfür werden zwei Linien aus der Grafik-Library verwendet, auf die ich im nächsten Kapitel ausführlich zu sprechen komme. Nehmen Sie zunächst einmal einfach hin, daß man mit der Draw-Funktion eine Linie zeichnen und mit der Move-Funktion den Grafikcursor positionieren kann.

Diese Routine ist relativ kurz und plottet dennoch ohne Absturzgefahr durch Bereichsüberschreitung eine schöne Grafik auf den Bildschirm. Ein Problem möchte ich Ihnen jedoch nicht verschweigen: Wenn Funktionen mit Asymptoten auftreten (z.B. 1/x, tan(x), etc.),

kann man das Pech haben, daß im Anpassungs-Maßstab die letzten berechneten Punkte von beiden Seiten der Asymptote noch gezeichnet werden. Diese werden dann illegalerweise durch eine Linie verbunden. Abhelfen kann man nur mit wesentlich mehr Programmieraufwand, indem man z.B. prüft, ob innerhalb der letzten beiden Punkte ein Vorzeichenwechsel stattgefunden hat. Man müßte in einem solchen Fall nach dem Vorzeichenwechsel den Grafik-Cursor wieder neu positionieren.

3.8.10 Öffnen der Windows und Auswertung des Message-Ports

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * Window öffnen und Message auswerten
00000A54 mainwindow       CALLINT OpenWindow
00000A5E                  move.l d0,iwindowpointer

00000A64 iloop            move.l iwindowpointer,a0
00000A6A                  move.l 86(a0),a0
00000A6E                  CALLEXEC GetMsg
00000A76                  tst.l d0           ;keine Message
00000A78                  beq iloop
00000A7C                  move.l d0,a2
00000A7E                  move.l $14(a2),d2  ;Wert holen:
00000A82                  cmpi.l #$200,d2    ;Close-Gadget
00000A88                  beq close
00000A8C                  move.l d0,a2
00000A8E                  move.l $1C(a2),a2  ;Adresse User-Gadget
00000A92                  move.w $26(a2),d2  ;Gadget-ID aus Gadget
00000A96                  cmpi.w #111,d2     ;Struktur holen
00000A9A                  beq close
00000A9E                  cmpi.w #112,d2
00000AA2                  beq close
00000AA6                  cmpi.w #113,d2
00000AAA                  bne iloop

00000AAE close            rts
```

Diese Universal-Routine wird von allen Menüs außer der Grafikausgabe aufgerufen. Zunächst wird ein Window geöffnet, dessen Struktur-Adresse im Register A0 übergeben wird. Dann kommt die Auswertung einer Message: Sie sehen, daß ich hier den Weg des Pollings gegangen bin, d.h., es wird so lange der Message-Port abgefragt, bis eine Message anliegt. Dieser Weg ist zwar nicht so schön wie der obige mit der Wait-Funktion, ich wollte Ihnen aber zeigen, daß auch dieses Verfahren funktioniert. Falls endlich eine Message vorliegt, wird zunächst das IDCMP-Flag geprüft. In Frage kommt das Close-

Gadget ($200), worauf sofort beendet wird, oder ein Gadget-Flag. Tritt dieser Fall ein, wird zunächst die Adresse der Gadget-Struktur und anschließend die ID-Nummer des Gadgets geholt, wie wir es in Kapitel 2 behandelt haben. Um zwischen den uns interessierenden Gadgets (z.B. Ende, Scrolling) und den übrigen, die die Eingabe betreffen, zu unterscheiden, habe ich den relevanten Gadgets die Nummern 111, 112 und 113 gegeben. Wird nun eines dieser Gadgets angewählt, wird ins Hauptprogramm zurückgesprungen, sonst kann man in Ruhe die nächste Message abwarten.

3.8.11 Umwandlung der Eingabestrings in FFP-Zahlen

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * Umwandlung String - Fliesskomma
00000AB0 stringfloat     clr.l float
00000AB6                 clr.w EXP
00000ABC                 clr.l d0
00000ABE                 clr.l d3
00000AC0                 move.l (a1),a5     ;Start Stringstruktur
00000AC2                 adda.l #16,a5
00000AC8                 clr.l d5
00000ACA                 move.w (a5),d5     ;Länge Text holen
00000ACC                 move.b (a2),d0     ;erstes Zeichen
00000ACE                 cmpi.b #"-",d0     ;Minuszeichen?
00000AD2                 bne plus
         * Negative Zahl

00000AD6              move.b #128,EXP       ;Negativ-Flag setzen
00000ADE              move.w d5,d0
00000AE0              move.l a2,a0          ;und Rest-String
00000AE2 strloop1     move.b 1(a0),(a0)+
00000AE6              subq #1,d0            ;um eine Position nach
00000AE8              bne strloop1
00000AEC              subq #1,d5            ;links schieben

00000AEE plus         clr.l d0              ;Position
00000AF0              move.l a2,a0
00000AF2 strloop2     move.b (a0)+,d3       ;Zeichen holen
00000AF4              cmpi.b #".",d3
00000AF8              beq found             ;Punkt gefunden
00000AFC              addq #1,d0
00000AFE              cmp.w d0,d5           ;alles durchsucht
00000B00              beq nopoint
00000B04              bra  strloop2

00000B08 found        subq #1,d0
00000B0A              move.l d0,d4          ;höchster Exponent
```

```
00000B0C              addq #1,d0
00000B0E              move.l a2,a0          ;Punkt aus String
00000B10 strloop3     move.l a0,a6
00000B12              adda.l d0,a6          ;eliminieren
00000B14              move.b 1(a6),(a6)
00000B18              addq #1,d0
00000B1A              cmp.w d0,d5
00000B1C              bne strloop3
00000B20              subq #1,d5
00000B22              bra  weiter1

00000B26 nopoint      move.l d5,d4
00000B28              subq #1,d4            ;höchster Exponent

00000B2A weiter1      clr.l d6              ;Position
00000B2C              move.l a2,a5

00000B2E strloop4     move.l d4,d0          ;Exponenten holen
00000B30              CALLFFP SPFlt         ;in Fließkomma
00000B3A              move.l d0,d1
00000B3C              move.l stellentab,d0 ;10 in Fließkomma
00000B42              CALLMATHTRANS SPPow  ;10 hoch x rechnen
00000B4C              move.l a5,a6          ;Faktor holen (ASCII)
00000B4E              adda.l d6,a6
00000B50              clr.l d1
00000B52              move.b (a6),d1
00000B54              subi.b #"0",d1        ;minus 48 ("0") =
00000B58              move.l d0,d7          ;Zahl in Fließkomma
00000B5A              move.l d1,d0          ;mit 10 hoch x
00000B5C              CALLFFP SPFlt
00000B66              move.l d7,d1
00000B68              CALLFFP SPMul         ;Multiplizieren
00000B72              move.l float,d1
00000B78              CALLFFP SPAdd         ;zu bisheriger Summe
00000B82              move.l d0,float       ;addieren
00000B88              subq.l #1,d4          ;Exponenten verringern
00000B8A              addq #1,d6            ;Position erhöhen
00000B8C              cmp.l d6,d5           ;fertig?
00000B8E              bne strloop4
00000B92              move.l float,d0       ;Negativ-Flag einblenden
00000B98              or.b EXP,d0
00000B9E              move.l d0,(a3)
00000BA0              move.l (a1)+,d0       ;und Wert in Tabelle
00000BA2              adda.l #16,a2
00000BA8              move.l (a3)+,d0
00000BAA              rts
```

```
00000BAC float        dc.l 0
00000BB0 EXP          dc.w 0

         * Tabelle mit Stringanfängen

00000BB2 strinfotabfak  dc.l strinfo1,strinfo2,strinfo3,strinfo4
00000BC2                dc.l strinfo5,strinfo6,strinfo7,strinfo8

00000BD2 strinfotabexp  dc.l strinfo11,strinfo12,strinfo12,strinfo14
00000BE2                dc.l strinfo15,strinfo16,strinfo17,strinfo18

00000BF2 strinfotabpar  dc.l strinfo20,strinfo21,strinfo22


         * Wandelt Faktoren in Fließkomma

00000BFE wandelfak       lea strinfotabfak,a1  ;Tabelle mit Anfängen
00000C04                 move.w #8,d2          ;Anzahl Gadgets
00000C08                 lea fakx1,a2          ;Start Strings
00000C0E strloop5        bsr  stringfloat
00000C12                 subq.w #1,d2
00000C14                 bne strloop5
00000C18                 rts

         * Wandelt Exponenten in Fließkomma

00000C1A wandelexp       lea strinfotabexp,a1  ;Tabelle mit Anfängen
00000C20                 move.w #8,d2          ;Anzahl Gadgets
00000C24                 lea expx1,a2          ;Start Strings
00000C2A strloop6        bsr  stringfloat
00000C2E                 subq.w #1,d2
00000C30                 bne strloop6
00000C34                 rts

         * Wandelt Parameter in Fließkomma

00000C36 wandelpar       lea strinfotabpar,a1  ;Tabelle mit Anfängen
00000C3C                 move.w #3,d2          ;Anzahl Gadgets
00000C40                 lea Untere,a2         ;Start Strings
00000C46 strloop7        bsr  stringfloat
00000C4A                 subq.w #1,d2
00000C4C                 bne strloop7
00000C50                 rts
```

Die Funktion dieser Umwandlungsroutine vom String- ins FFP-Format kennen Sie bereits aus dem letzten Abschnitt, liebe Leser. Daher möchte ich Sie nicht mit einer Wiederholung der Funktionsbeschreibung langweilen, sondern nur auf den Aufruf der diversen Unterprogramme eingehen.

Zwei Dinge benötigt die Routine, um korrekt arbeiten zu können: Zum einen natürlich den Zeiger auf den Textpuffer, in dem der umzuwandelnde String steht. Zum anderen aber auch einen Zeiger auf die Stringinfo-Struktur des Gadgets, da dort die Länge des Eingabetextes vermerkt ist. Um nicht für jedes Gadget neue Parameterübergaben veranstalten zu müssen, wurden die Startadressen der String-Info-Strukturen in einer Tabelle abgelegt. Da immer jeweils acht Stringgadgets nacheinander abgearbeitet werden (entweder acht Faktoren oder Exponenten), braucht so nur der Startwert der Tabelle übergeben zu werden. Die Routine erhöht dann in jedem Durchlauf automatisch sowohl den Zeiger auf die String-Info-Strukturen als auch denjenigen auf das Texteingabefeld, da auch hier der Abstand von einem Feld zum nächsten konstant bekannt ist.

Durch dieses Verfahren kann die Routine sowohl für Faktoren, Exponenten und Parameter genutzt werden. Sie sehen, daß sich die einzelnen Aufrufsequenzen nur in den übergebenen Parametern in A1 (Zeiger auf Stringinfo-Tabelle) und A2 (Start der Texteingabefelder) unterscheiden.

3.8.12 Funktionswertberechnung

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * Berechnungen. Daten ab xwert und ywert
00000C52 berechnung   move.l #1,d6            ;Anzahl Wertepaare
00000C58              move.l floatuntere,d7   ;aktueller x-Wert
00000C5E              lea xwert,a3

00000C64 rechenloop   bsr  berechnez          ;Zähler berechnen
00000C68              move.l float,d2         ;Ergebnis merken
00000C6E              move.l float,8000(a3)
00000C76              bsr  berechnen          ;Nenner berechnen
00000C7A              move.l float,d0
00000C80              move.l float,12000(a3)
00000C88              clr.l d1                ;Mit 0 vergleichen
00000C8A              CALLFFP SPCmp
00000C94              tst.l d0                ;wenn Nenner = 0,
00000C96              bne ungleich
00000C9A              move.l #$FFFFFFFF,d0    ;Ergebnis =
00000CA0              bra  divbyzero          ;unendlich
00000CA4 ungleich     move.l d2,d0
00000CA6              move.l float,d1
00000CAC              CALLFFP SPDiv           ;Zähler/Nenner
00000CB6              bsr  round              ;runden
```

```
00000CBA divbyzero    move.l d0,4000(a3)      ;nach y-wert
00000CBE              move.l d7,d0
00000CC0              bsr  round              ;runden
00000CC4              move.l d0,(a3)+         ;nach x-wert
00000CC6              move.l d7,d0
00000CC8              move.l floatschritt,d1  ;schrittweite
00000CCE              CALLFFP SPAdd           ;hinzuaddieren
00000CD8              move.l d0,d7            ;ist neues x
00000CDA              move.l floatobere,d1    ;obere Grenze erreicht?
00000CE0              CALLFFP SPCmp
00000CEA              cmpi.l #-1,d0
00000CF0              beq rechenend           ;ja, fertig
00000CF4              addq #1,d6              ;Zähler erhöhen
00000CF6              bra  rechenloop

00000CFA rechenend    move.w d6,paarzahl
00000D00              rts

00000D02 paarzahl     dc.w 0
                      cnop 0,4

         *Werte runden nach y=int(y*1000|100|10+0.5)/1000|100|10

00000D04 round      clr.l d2
00000D06            move.b stellen,d2
00000D0C            subq #1,d2                ;Nachkommastellen -1
00000D0E            lsl.l #2,d2               ;mal 4 wegen Langwort
00000D10            addi.l #stellentab,d2     ;Plus Tabellenbasis
00000D16            move.l d2,a0
00000D18            move.l (a0),d2            ;Wert holen (1000,100,10)
00000D1A            move.l d2,d1
00000D1C            CALLFFP SPMul             ;Multiplizieren
00000D26            move.l #$80000040,d1      ;0.5 addieren
00000D2C            btst.l #7,d0              ;Negativ-Flag gesetzt?
00000D30            beq roundplus             ;nein
00000D34            ori.l #%00000000000000000000000010000000,d1
00000D3A roundplus  CALLFFP SPAdd
00000D44            CALLFFP SPFix             ;nach Integer
00000D4E            CALLFFP SPFlt             ;nach Fließkomma
00000D58            move.l d2,d1
00000D5A            CALLFFP SPDiv             ;durch 1000/100/10 dividieren
00000D64            rts

00000D66 stellentab dc.b $A0,$00,$00,$44      ;10
00000D6A            dc.b $C8,$00,$00,$47      ;100
00000D6E            dc.b $FA,$00,$00,$4A      ;1000
```

```
* Berechnung des Polynoms
* Durch DOS-Fehler länger als notwendig
* Gleicher Fehler wie beim C-64! (z.B. -2 hoch 3=-8, DOS sagt: +8)


00000D72 polynom    clr.l float              ;Summe löschen
00000D78            move.l #8,d3             ;Zähler setzen
00000D7E polyloop1  move.l d7,d0             ;x-Wert holen
00000D80            clr.l d1                 ;kleiner null
00000D82            CALLFFP SPCmp
00000D8C            cmpi.l #1,d0
00000D92            bne ok                   ;nein, alles klar
00000D96            move.l #2,d0
00000D9C            CALLFFP SPFlt            ;Zahl 2 in Fließkomma
00000DA6            move.l d0,d1
00000DA8            move.l (a2),d0           ;Exponenten holen
00000DAA            CALLFFP SPDiv            ;durch 2 teilen
00000DB4            move.l d0,a5
00000DB6            CALLFFP SPFix            ;in Integer wandeln
00000DC0            CALLFFP SPFlt            ;wieder in Fließkomma
00000DCA            move.l a5,d1
00000DCC            CALLFFP SPCmp            ;beide Zahlen gleich?
00000DD6            tst.l d0                 ;ja, dann gerade Zahl
00000DD8            beq ok
00000DDC            move.l #$80,a5
00000DE2            bra  neg
00000DE6 ok         move.l #0,a5
00000DEC neg        move.l d7,d0             ;x-Wert holen
00000DEE            move.l (a2)+,d1          ;Exponenten holen
00000DF0            CALLMATHTRANS SPPow      ;x hoch berechnen
00000DFA            move.l a5,d1             ;eventuelles Negativ-
00000DFC            or.l d1,d0               ;Flag einblenden
00000DFE            move.l (a1)+,d1          ;Faktor holen
00000E00            CALLFFP SPMul            ;Multiplizieren
00000E0A            move.l float,d1          ;Summe holen
00000E10            CALLFFP SPAdd            ;Wert hinzuaddieren
00000E1A            move.l d0,float          ;Summe speichern
00000E20            subq #1,d3
00000E22            bne polyloop1
00000E26            rts

         *Zaehler berechnen

00000E28 berechnez  lea floatexpz,a2
00000E2E            lea floatfakz,a1
00000E34            bra  polynom
```

```
        *Nenner berechnen

00000E38 berechnen    lea floatexpn,a2
00000E3E              lea floatfakn,a1
00000E44              bra  polynom
```

Endlich haben wir das »Herz« des Programms, die Berechnungsroutine erreicht! Das Hauptprogramm beginnt beim Label »berechnung« und leistet folgendes: Zunächst wird der x-Startwert auf die untere Grenze gesetzt und der Zeiger für die Ergebnisse auf »xwert« gesetzt. Es ist maximal Platz für 1000 Werte geschaffen worden, was lässig ausreichen sollte. Nach 4000 Byte (1000 FFP-Langwörter) folgt der Speicher für die Funktionsergebnisse, nach jeweils weiteren 4000 Byte der Speicher für die Werte des Zähler- und Nennerpolynoms allein. Zähler- und Nennerresultate werden in den Routinen »berechnez« und »berechnen« berechnet und in der Variablen »float« abgespeichert. Von dort werden sie dann in die Arrays übertragen. Nun wird der Nennerwert mit Null verglichen. Dann ist eine Division nämlich unzulässig und würde endlich einmal wieder eine Tätigkeit des Gurus hervorrufen. Da wir ihn aber nicht stören wollen, prüfen wir lieber auf Null und legen in diesem Fall das Ergebnis als $FFFFFFFF fest, was uns später bei der Ausgabe ein Anhaltspunkt sein wird. Sonst dividieren wir Zähler durch Nenner und runden das Ergebnis im Unterprogramm »round« gemäß der festgelegten Anzahl der Nachkommastellen. Für eine Rundung gilt:

1	Nachkommastelle	: wert = INT(wert*10+0.5)/10
2	Nachkommastellen	: wert = INT(wert*100+0.5)/100
3	Nachkommastellen	: wert = INT(wert*1000+0.5)/1000

Obwohl dieses Verfahren schon sehr alt ist, schütteln immer noch viele Menschen den Kopf über das Funktionsprinzip. Daher möchte ich es kurz an einem Beispiel der Zahl 3.6787 erläutern, die auf drei Stellen gerundet werden soll: Im ersten Schritt wird die Zahl mit 1000 multipliziert. Dies bedeutet nichts anderes, als daß das Komma um drei Stellen nach rechts verschoben wird. Der Hintergedanke dabei ist, daß damit die ersten drei Nachkommastellen vor dem Abschneiden durch die Integer-Funktion gerettet werden:

3.6787*1000 = 3678.7

Nun muß man noch 0.5 addieren, damit ein Wert der ersten Nachkommastelle, die ja nun für die Rundung relevant ist, der größer als 0.5 ist, überhaupt noch in dem Vorkommawert berücksichtigt werden kann:

3678.7+0.5 = 3679.2

Durch die SPFix-Funktion werden alle Nachkommastellen radikal weggeputzt, so daß der Wert

3679

übrigbleibt. Wenn man ihn durch 1000 teilt, das Komma also wieder um drei Stellen nach links verschiebt, ergibt sich unser (korrekt) gerundeter Wert von

3.679

Bei negativen Zahlen ist zu beachten, daß man nicht 0.5 addieren, sondern subtrahieren muß. Deshalb prüft unsere Routine auf einen negativen Wert und blendet bei Bedarf in die FFP-Zahl 0.5 ein Negativ-Bit ein.

Der berechnete Wert kann jetzt in das Array eingetragen werden. Dann wird der x-Wert um den Wert der festgelegten Schrittweite erhöht und darauf überprüft, ob er die obere Grenze überschritten hat. Solange dies nicht der Fall ist, werden in der Schleife weitere Werte berechnet.

Die Berechnung des Polynoms wäre an sich trivial, wenn da nicht der oben schon erwähnte DOS-Fehler bei der Potenzierung wäre. Er wurde natürlich in unserer Routine »polynom« berücksichtigt, deren Aufgabe darin besteht, die Summe der acht Potenzen (Faktor*10'Exponent) zu berechnen. Dafür werden ihr von den aufrufenden Programmen »berechnez« und »berechnen« die Startadressen der Felder übergeben, in denen sich die ins FFP-Format umgewandelten Faktoren und Exponenten befinden.

3.8.13 Ausgabe eines Wertepaares auf Bildschirm oder Drucker

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         *Gibt ein x,y-Paar auf dem Bildschirm aus
00000E48 xyout          bsr  clearstring       ;String vorbereiten
00000E4C                cmpi.l #$FFFFFFFF,d3   ;unendlich?
00000E52                bne vernünftig         ;nein
00000E56                move.l #11,d0
00000E5C                lea textstring1+11,a0
00000E62                lea unendlichstr,a1
00000E68 unloop         move.b (a1)+,(a0)+
00000E6A                subq #1,d0
00000E6C                bne unloop
00000E70                bra  p4

00000E74 unendlichstr   dc.b "Unendlich",0
                        even


00000E7E vernünftig  lea textstring1,a0
00000E84             cmpi.b #$80,d3            ;negative Zahl?
00000E88             bmi positiv
00000E8C             move.b #"-",(a0)          ;ja, Minuszeichen einfügen
00000E90             subi.b #$80,d3
00000E94 positiv     move.l #1,d2              ;Durchgang
```

```
00000E9A        move.l #zehnhochxtab,d6   ;Tabelle mit Potenzen
00000EA0        move.l #23,d5             ;Anzahl der Nachkommastellen
00000EA6 w3     move.l #19,d0             ;höchster Exponent
00000EAC        sub.l d2,d0
00000EAE        addq.l #3,d0              ;Offset
00000EB0        lsl.l #2,d0               ;mal 4 wegen Langwort
00000EB2        add.l d6,d0               ;plus Basis der Tabelle
00000EB4        move.l d0,a5
00000EB6        move.l (a5),d0            ;Wert holen
00000EB8        move.l d0,d7
00000EBA w2     move.l d7,d0
00000EBC        move.l d3,d1              ;mit Zahl vergleichen
00000EBE        CALLFFP SPCmp
00000EC8        cmpi.l #-1,d0
00000ECE        beq next                  ;10 hoch x größer
00000ED2        move.l #textstring1,d0
00000ED8        add.l d2,d0
00000EDA        cmpi.l #20,d2
00000EE0        blt w1
00000EE4        addq #1,d0                ;Wert der Position im
00000EE6 w1     move.l d0,a0              ;String um eins
00000EE8        clr.l d0
00000EEA        move.b (a0),d0            ;erhöhen
00000EEC        addq #1,d0
00000EEE        move.b d0,(a0)
00000EF0        move.l d3,d0
00000EF2        move.l d7,d1              ;Zahl verringern um
00000EF4        CALLFFP SPSub
00000EFE        move.l d0,d3              ;10 hoch x
00000F00        bra  w2

00000F04 next   addq #1,d2                ;Zähler erhöhen
00000F06        cmp.l d5,d2
00000F08        bne w3

         *Überflüssige Nachkommastellen abschneiden

00000F0C        move.l #20,d0
00000F12        add.b stellen,d0          ;Nachkommastellen addieren
00000F18 p1     move.l #textstring1,d1
00000F1E        add.l d0,d1
00000F20        move.l d1,a0              ;wenn "0" gefunden,
00000F22        move.b (a0),d2
00000F24        tst.b d2                  ;eliminieren
00000F26        bne nonull
00000F2A        subq #1,d0
00000F2C        cmpi.l #20,d0
```

```
00000F32              bne p1
00000F36              clr.b -(a0)          ;Keine Nachkommastellen
00000F38              bra  p4
00000F3C nonull       clr.b 1(a0)

          * Überflüssige Nullen am Anfang durch Leerzeichen ersetzen

00000F40 p4           move.l #textstring1,a0
00000F46              adda.l #1,a0
00000F4C p2           move.b (a0)+,d3      ;Zeichen holen
00000F4E              cmpi.b #"0",d3       ;"0"?
00000F52              bne pfertig          ;nein, fertig
00000F56              move.b (a0),d0
00000F58              cmpi.b #".",d0       ;"."?
00000F5C              beq pfertig          ;ja, fertig
00000F60              tst.b d0
00000F62              beq pfertig          ;Leerzeichen einfügen
00000F66              move.b -2(a0),-1(a0)
00000F6C              move.b #" ",-2(a0)
00000F72              bra  p2

00000F76 pfertig      rts

          * Potenzen im Fließkommaformat

                         even
00000F78 zehnhochxtab    dc.b $83,$12,$72,$37    ;10 hoch -3
00000F7C                 dc.b $A3,$D7,$0A,$3A    ;10 hoch -2
00000F80                 dc.b $CC,$CC,$CC,$3D    ;10 hoch -1
00000F84                 dc.b $80,$00,$00,$41    ;10 hoch 0
00000F88                 dc.b $A0,$00,$00,$44    ;10 hoch 1
00000F8C                 dc.b $C8,$00,$00,$47    ;10 hoch 2
00000F90                 dc.b $FA,$00,$00,$4A    ;10 hoch 3
00000F94                 dc.b $9C,$40,$00,$4E    ;10 hoch 4
00000F98                 dc.b $C3,$50,$00,$51    ;10 hoch 5
00000F9C                 dc.b $F4,$24,$00,$54    ;10 hoch 6
00000FA0                 dc.b $98,$96,$80,$58    ;10 hoch 7
00000FA4                 dc.b $BE,$BC,$20,$5B    ;10 hoch 8
00000FA8                 dc.b $EE,$6B,$28,$5E    ;10 hoch 9
00000FAC                 dc.b $95,$02,$F9,$62    ;10 hoch 10
00000FB0                 dc.b $BA,$43,$B7,$65    ;10 hoch 11
00000FB4                 dc.b $E8,$D4,$95,$68    ;10 hoch 12
00000FB8                 dc.b $91,$84,$D1,$6C    ;10 hoch 13
00000FBC                 dc.b $B5,$E5,$F7,$6F    ;10 hoch 14
00000FC0                 dc.b $E3,$5F,$83,$72    ;10 hoch 15
00000FC4                 dc.b $8E,$1B,$A7,$76    ;10 hoch 16
00000FC8                 dc.b $B1,$A2,$9B,$79    ;10 hoch 17
```

```
00000FCC                  dc.b $DE,$0B,$50,$7C    ;10 hoch 18
                 cnop 0,4


         *x-Wert auf Bildschirm

00000FD0 xout             bsr  xyout
00000FD4                  bra  ausx

         *y-Wert auf Bildschirm

00000FD8 yout             bsr  xyout
00000FDC                  bra  ausy

         * Ausgabestring vorbereiten

00000FE0 clearstring      lea textstring1,a0
00000FE6                  move.b #" ",(a0)+
00000FEA                  move.l #23,d0           ;23 Nullen schreiben
00000FF0 clearloop        move.b #"0",(a0)+
00000FF4                  subq #1,d0
00000FF6                  bne clearloop
00000FFA                  lea textstring1,a0
00001000                  move.b #".",20(a0)      ;Punkt setzen
00001006                  rts


         *x-Wert auf Drucker vorbereiten

00001008 xdruck           move.l #xstring,a0      ;String löschen
0000100E                  adda.l #1,a0
00001014                  move.l a0,a1            ;24 Leerzeichen schreiben
00001016                  move.l #24,d0
0000101C xd1              move.b #" ",(a0)+
00001020                  subq #1,d0
00001022                  bne xd1
00001026                  bsr  xyout              ;String generieren
0000102A                  lea textstring1,a0
00001030 xd3              move.b (a0)+,d0         ;Zeichen übernehmen
00001032                  beq xd2
00001036                  move.b d0,(a1)+
00001038                  bra  xd3
0000103C xd2              rts

         *y-Wert auf Drucker vorbereiten

0000103E ydruck           lea ystring,a0          ;String löschen
```

```
00001044                move.l #24,d0
0000104A yd1            move.b #" ",(a0)+       ;24 Leerzeichen schreiben
0000104E                subq #1,d0
00001050                bne yd1
00001054                bsr  xyout              ;String generieren
00001058                lea textstring1,a0
0000105E                lea ystring,a1
00001064 yd3            move.b (a0)+,d0         ;Zeichen übernehmen
00001066                beq yd2
0000106A                move.b d0,(a1)+
0000106C                bra  yd3
00001070 yd2            rts


         * Wertepaar ausdrucken

00001072 druck          move.l printerpointer,d1
00001078                move.l #xstring,d2
0000107E                move.l #56,d3           ;Stringlänge
00001084                CALLDOS Write
0000108E                rts
                        even

00001090 xstring   dc.b "/",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
000010A9           dc.b "  /"
000010AC ystring   dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"/",
                   $0A


                         even

         *Überschrift ausdrucken

000010C8 über            move.l printerpointer,d1
000010CE                 move.l #überstring,d2
000010D4                 move.l #112,d3
000010DA                 CALLDOS Write
000010E4                 rts

000010E6 überstring  dc.b "/           X             /            Y  /",$0A
0000111E  dc.b "/-------------------------/-------------------------/",$0A

                         even
```

Die Umwandlung vom Fließkomma- ins String-Format haben wir ebenfalls schon ausführlich besprochen. Das einzige Ihnen nicht Vertraute wird die Prüfung des Wertes auf $FFFFFFFF sein. Diesen Wert haben wir oben als Platzhalter für unendlich definiert. Darum wird bei Auftreten dieses Wertes das Wort »Unendlich« in den Textpuffer geschrieben und mit der Umwandlung abgebrochen. Es wirkt wirklich professionell, wenn statt eines Gurus oder einer extrem hohen Zahl das ausgeschriebene Wort »Unendlich« auf dem Bildschirm oder Drucker erscheint. Leider ist es aber mit der Rechengenauigkeit unseres Amiga nicht so weit her, daß man annehmen könnte, eine Plotstelle könne man mit Sicherheit ausmachen. Falls z.B. die Funktion (1/x) im Bereich von −2 bis +2 mit einer Schrittweite von 0.5 untersucht werden soll, müßte man theoretisch als fünften x-Wert eine 0 und damit als Funktionswert »Unendlich« herausbekommen. Leider sind die Additionen mit einer schon ungenauen Schrittweite so ungenau, daß man nicht sicher sein kann, wirklich die Null zu treffen.

Die Funktion des Unterprogramms »clearstring« kennen Sie auch schon: es initialisiert den Textspeicher für die Umwandlung. Die Routinen »xdruck« und »ydruck« bereiten die Druckerausgabe vor, indem Sie den erzeugten String nach der Umwandlung in einen speziellen Ausgabestring für den Druck hineinkopieren. Wenn Sie sich die Texte »xstring« und »ystring« ansehen, erkennen Sie, daß die Tabelle offenbar durch Schrägstriche eingerahmt wird. In diese Strings wird die Zahl im Stringformat hineingeschrieben. In der anschließenden Routine »druck« werden beide Strings in einer Zeile ausgedruckt, wofür die DOS-Funktion Write zuständig ist. Ihr wird im Register D3 die Gesamtlänge der auszugebenden Zeichen (hier also Länge xstring + ystring) sowie im Register D2 der Zeiger auf den String übergeben. Das Unterprogramm »ueber« schließlich druckt noch eine Kopfzeile für die Tabelle, die in dem String ab »ueberstring« gespeichert ist.

3.8.14 Programmende

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * ENDE MENÜPUNKTE
00001156 exit2   move.l iwindowpointer,a0          ;Window schließen
0000115C         CALLINT CloseWindow
00001166         move.l plotpointer,a0             ;Screen schließen
0000116C         CALLINT CloseScreen
00001176         bra  start                        ;von vorne

0000117A exit    move.l iwindowpointer,a0
00001180         CALLINT CloseWindow
0000118A exit1   bra  start


         *E N D E   P R O G R A M M

0000118E ende    move.l windowpointer,a0           ;Menü löschen
```

```
00001194          CALLINT ClearMenuStrip

0000119E          move.l windowpointer,a0       ;Window schließen
000011A4          CALLINT CloseWindow
000011AE          move.l screenpointer,a0       ;Screen schließen
000011B4          CALLINT CloseScreen

000011BE          move.l _DOSBase,a1            ;Alle Libraries schließen
000011C4          CALLEXEC CloseLibrary
000011CC fini5    move.l _GfxBase,a1
000011D2          CALLEXEC CloseLibrary
000011DA fini4    move.l _MathTransBase,a1
000011E0          CALLEXEC CloseLibrary
000011E8 fini3    move.l _IntuitionBase,a1
000011EE          CALLEXEC CloseLibrary
000011F6 fini2    move.l _MathBase,a1
000011FC          CALLEXEC CloseLibrary
00001204 fini1    rts

                  cnop 0,2


00001206 _DOSBase  dc.l 0                       ;Basis-Pointer Libraries
0000120A _MathBase dc.l 0
0000120E _IntuitionBase  dc.l 0
00001212 _MathTransBase  dc.l 0
00001216 _GfxBase dc.l 0

0000121A dosname  dc.b "dos.library",0          ;Namen Libraries cnop 0,2
00001226 grafname  dc.b "graphics.library",0
00001237          cnop 0,2
00001238 ffpname  dc.b "mathffp.library",0 cnop 0,2
00001248 intname  dc.b "intuition.library",0 cnop 0,2
0000125A mathtransname dc.b "mathtrans.library",0 cnop 0,2
```

Das Programmende ist erreicht, liebe Leser ! Die Label »exit2« und »exit1« werden nach jedem beendeten Menüpunkt angesprungen, wobei »exit2« nur nach der Grafikausgabe angesprungen wird, da dort zusätzlich ein Screen geschlossen werden muß. Nachdem auch das Window geschlossen wurde, wird wieder an den Anfang des Programms gesprungen, wo auf eine erneute Anwahl eines Menüpunktes gewartet wird. Wurde der Menüpunkt »Ende« angewählt, wird das Menü gelöscht und alle Libraries geschlossen. Dann wird mit dem RTS-Befehl in das CLI bzw. in die Workbench zurückgesprungen.

3.8.15 Die Strukturen

HiSoft GenAmiga Assembler 1.21 »Wertetabelle«

```
         * Strukturen
         * a) SCREENS
0000126C screen_defs     SCREEN 320,200,2,screentitel
0000128C plot_defs       SCREEN 640,400,$8000,plottitel

000012AC screentitel  dc.b "Wertetabelle",0
000012B9         cnop 0,4

000012BC plottitel   dc.b "Funktion plotten",0
000012CD         cnop 0,4

000012D0 screenpointer  dc.l 0
000012D4 plotpointer     dc.l 0

         * b) WINDOWS

000012D8 zaehlwindow WINDOW 0,20,280,180,$260,igadget1, zwindowname,
         point1,$1006
00001308 nennwindow  WINDOW 0,20,280,180,$260,igadget1,nwindowname,
         point2,$1006
00001338 parawindow  WINDOW 0,20,280,180,$260,igadget20,pwindowname,
         point3,$1006
00001368 auswindow   WINDOW 0,20,320,180,$260,igadget30,awindowname,
         point4,$1006
00001398 startwindow WINDOW 60,60,200,80,$100,0,swindowname,
         point5,$1006
000013C8 plotwindow  WINDOW 0,0,640,256,$200,0,plotwindowname,
         point6,$100E

000013F8 zwindowname  dc.b   "Funktionseingabe Zaehler",0
00001411         cnop 0,2
00001412 nwindowname  dc.b   "Funktionseingabe Nenner",0
                 cnop 0,2
0000142A pwindowname  dc.b   "Parametereingabe",0
0000143B         cnop 0,2
0000143C awindowname  dc.b   "         x  Funktionswerte      y",0
                 cnop 0,2
0000145E swindowname  dc.b   "Wertetabelle Amiga",0
00001471         cnop 0,4
00001474 plotwindowname  dc.b "Grafische Funktionsdarstellung",0
00001493         cnop 0,4
```

```
00001494 iwindowpointer  dc.l 0
00001498 windowpointer dc.l 0

                         even

         * c) TEXTE
         * Ausgabe x,y-Werte

0000149C ausx            move.l #2,d0
000014A2                 lea textstringx,a1
000014A8                 bra  ausweiter

000014AC ausy            move.l #125,d0
000014B2                 lea textstringy,a1

000014B8 ausweiter       clr.l d1
000014BA                 move.b zeile,d1
000014C0                 move.l iwindowpointer,a0
000014C6                 move.l 50(a0),a0
000014CA                 CALLINT PrintIText
000014D4                 rts

000014D6 textstringx     TEXTOUT textstring1+8
000014EA textstringy     TEXTOUT textstring1

000014FE textstring1     dc.b " 0000000000000000000.000",0,0,0
                         even

0000151A zeile           dc.l 0


0000151E error           TEXTOUT1 20,7,errortop
00001532 error1          TEXTOUT1 8,4,errorleft
00001546 error2          TEXTOUT1 8,4,errorright

0000155A errortop        dc.b "Ein- Ausgabefehler",0
                         even
0000156E errorleft       dc.b "Nochmal",0
                         even
00001576 errorright      dc.b "Abbruch",0
                         even

         *Requester aufrufen

0000157E requester       move.l windowpointer,a0
00001584                 lea error,a1
0000158A                 lea error1,a2
```

```
00001590                  lea error2,a3
00001596                  clr.l d0
00001598                  clr.l d1
0000159A                  move.l #200,d2
000015A0                  move.l #55,d3
000015A6                  CALLINT AutoRequest
000015B0                  rts


000015B2 print            PRINTOUT text1,30,20
000015D8                  PRINTOUT text2,30,40
000015FE                  PRINTOUT text3,30,60
00001624                  rts


00001626 text1            TEXTOUT text11
0000163A text2            TEXTOUT text22
0000164E text3            TEXTOUT text33

                          even
00001662 text11           dc.b "Wertetabelle V1.0",0
                          even
00001674 text22           dc.b "   (w) 1988 by   ",0
                          even
00001686 text33           dc.b "F.Riemenschneider",0
                          even

         * d) MENÜS


00001698 menu          MENUE  menu1,10,0,titel0,eintrag000
000016B6 menu1         MENUE  menu2,80,0,titel1,eintrag100
000016D4 menu2         MENUE  0,160,0,titel2,eintrag200


000016F2 eintrag000    MENUPUNKT eintrag010,0,0,120,$56,0,text000,"Z",0
00001716 eintrag010    MENUPUNKT eintrag020,0,15,120,$56,0,text010,"N",0
0000173A eintrag020    MENUPUNKT 0,0,30,120,$56,0,text020,"P",0

0000175E eintrag100    MENUPUNKT eintrag110,0,0,100,$52,0, text100,0,submenu
00001782 eintrag110    MENUPUNKT eintrag120,0,15,100,$52,0,text110,0,submenu1
000017A6 eintrag120    MENUPUNKT 0,0,30,100,$56,0,text120,"E",0

000017CA eintrag200    MENUPUNKT eintrag210,0,0,120,$56,0,text200,"O",0
000017EE eintrag210    MENUPUNKT eintrag220,0,15,120,$56,0,text210,"U",0
00001812 eintrag220    MENUPUNKT 0,0,30,120,$56,0,text220,"F",0
00001836 submenu       MENUPUNKT submenu01,110,0,115,$56,0,texts00,"B",0
```

```
0000185A submenu01    MENUPUNKT 0,110,15,115,$56,0,texts01,"D",0

0000187E submenu1     MENUPUNKT submenu11,110,0,115,$157,$FFFE,texts10,"1",0
000018A2 submenu11    MENUPUNKT 0,110,15,115,$57,$FFFD,texts11,"A",0


                         even
000018C6 titel0          dc.b "Eingabe",0
                         even
000018CE titel1          dc.b "Option",0
                         even
000018D6 titel2          dc.b "Plotten",0
                         even

000018DE text000         TEXTOUT1 0,0,textt000
000018F2 text010         TEXTOUT1 0,0,textt010
00001906 text020         TEXTOUT1 0,0,textt020
0000191A text100         TEXTOUT1 0,0,textt100
0000192E text110         TEXTOUT1 0,0,textt110
00001942 text120         TEXTOUT1 0,0,textt120
00001956 text200         TEXTOUT1 0,0,textt200
0000196A text210         TEXTOUT1 0,0,textt210
0000197E text220         TEXTOUT1 0,0,textt220

00001992 texts00         TEXTOUT1 0,0,textts00
000019A6 texts01         TEXTOUT1 0,0,textts01
000019BA texts10         TEXTOUT1 0,0,textts10
000019CE texts11         TEXTOUT1 0,0,textts11


                         even
000019E2 textt000        dc.b "Zähler",0
                         even
000019EA textt010        dc.b "Nenner",0
                         even
000019F2 textt020        dc.b "Parameter",0
                         even
000019FC textt100        dc.b "Ausgabe",0
                         even
00001A04 textt110        dc.b "Maßstab",0
                         even
00001A0C textt120        dc.b "Ende",0
                         even
00001A12 textt200        dc.b "Zähler",0
                         even
00001A1A textt210        dc.b "Nenner",0
                         even
```

```
00001A22 textt220        dc.b "Funktion",0
                         even

00001A2C textts00        dc.b "Bildschirm",0
                         even
00001A38 textts01        dc.b "Drucker",0
                         even
00001A40 textts10        dc.b "  1 : 1",0
                         even
00001A48 textts11        dc.b "  Anpassung",0
                         even

         * e) GADGETS

         border
00001A54          dc.w 0,0           ;x,y-Offset
00001A58          dc.b 3,1           ;Farben
00001A5A          dc.b  0            ;Farbmodus
00001A5B          dc.b  5            ;Zahl Koordinatenpaare
00001A5C          dc.l  koord1       ;Koordinatenpointer
00001A60          dc.l  0            ;nächster Border

         koord1

00001A64         dc.w    -2,-2
00001A68         dc.w    122,-2
00001A6C         dc.w    122,9
00001A70         dc.w    -2,9
00001A74         dc.w    -2,-2

         regler
00001A78          dc.w 0,0           ;x,y-Offset
00001A7C          dc.w 16,7          ;16 x 7 Pixel groß
00001A80          dc.w  1            ;1 Bitplane
00001A82          dc.l reglerdata    ;Reglerdaten
00001A86          dc.b 1,0           ;Farben
00001A88          dc.l 0             ;nächstes Image
         reglerdata

00001A8C         dc.w %0000001110000000
00001A8E         dc.w %0000011111000000
00001A90         dc.w %0000111111100000
00001A92         dc.w %0001111111110000
00001A94         dc.w %0000001110000000
00001A96         dc.w %0000001110000000
00001A98         dc.w %0000001110000000
```

```
         propinfo

00001A9A         dc.w  2            ;Horizontal verschieblich
00001A9C         dc.w 0,0           ;x,y-Position
00001AA0         dc.w $ffff/3       ;Schrittweite=1/20
00001AA2         dc.w 0             ;keine vertikale Verschiebung
00001AA4         dc.w 0             ;Breite
00001AA6         dc.w 0             ;Höhe
00001AA8         dc.w 0             ;Schrittweite horizontal
00001AAA         dc.w 0             ;Schrittweite vertikal
00001AAC         dc.w 0             ;linker Rand
00001AAE         dc.w 0             ;rechter Rand



00001AB0 string11        dc.b  "x",0
                 even
00001AB2 string22        dc.b  " Untere Grenze :",0
                 even
00001AC4 string33        dc.b  " Obere Grenze :",0
                 even
00001AD6 string44        dc.b  " Schrittweite :",0
                 even
00001AE8 string55        dc.b  "Nachkommastelle:",0
                 even
00001AFA string66        dc.b  "Eingabe beenden",0
                 even
00001B0A string77        dc.b  "1      2      3",0
                 even

00001B1A string88        dc.b  "Scroll Aufwärts",0
                 even
00001B2A string99        dc.b  "Scroll Abwärts",0
                 even
00001B3A string100       dc.b  "Ausgabe beenden",0
                 even
00001B4A string1111  dc.b  "--------------------------------------",0
                 even


00001B72 string1         STRING 125,0,string11,0
00001B86 string2         STRING -133,0,string22,0
00001B9A string3         STRING -133,0,string33,0
00001BAE string4         STRING -133,0,string44,0
00001BC2 string5         STRING -133,0,string55,string7
00001BD6 string6         STRING 0,0,string66,0
00001BEA string7         STRING 0,-10,string77,0
```

```
00001BFE string8         STRING 0,0,string88,string111
00001C12 string9         STRING 0,0,string99,0
00001C26 string10        STRING 0,0,string100,0
00001C3A string111       STRING -2,-10,string1111,0

         * Faktoren

00001C4E igadget1        GADGET igadget2,10,20,4,border,string1,strinfo1,1
00001C7A igadget2        GADGET igadget3,10,35,4,border,string1,strinfo2,2
00001CA6 igadget3        GADGET igadget4,10,50,4,border,string1,strinfo3,3
00001CD2 igadget4        GADGET igadget5,10,65,4,border,string1,strinfo4,4
00001CFE igadget5        GADGET igadget6,10,80,4,border,string1,strinfo5,5
00001D2A igadget6        GADGET igadget7,10,95,4,border,string1,strinfo6,6
00001D56 igadget7        GADGET igadget8,10,110,4,border,string1,strinfo7,7
00001D82 igadget8        GADGET igadget11,10,125,4,border,string1,strinfo8,8

00001DAE strinfo1         STRINF fakx1
00001DD2 strinfo2         STRINF fakx1+16*1
00001DF6 strinfo3         STRINF fakx1+16*2
00001E1A strinfo4         STRINF fakx1+16*3
00001E3E strinfo5         STRINF fakx1+16*4
00001E62 strinfo6         STRINF fakx1+16*5
00001E86 strinfo7         STRINF fakx1+16*6
00001EAA strinfo8         STRINF fakx1+16*7

00001ECE fakx1          ZAHLENSTRINGS "1","1","1","1","1","1","1","1"

         * Exponenten
00001F4E igadget11       GADGET igadget12,150,20,4,border,0,strinfo11,11
00001F7A igadget12       GADGET igadget13,150,35,4,border,0,strinfo12,12
00001FA6 igadget13       GADGET igadget14,150,50,4,border,0,strinfo13,13
00001FD2 igadget14       GADGET igadget15,150,65,4,border,0,strinfo14,14
00001FFE igadget15       GADGET igadget16,150,80,4,border,0,strinfo15,15
0000202A igadget16       GADGET igadget17,150,95,4,border,0,strinfo16,16
00002056 igadget17       GADGET igadget18,150,110,4,border,0,strinfo17,17
00002082 igadget18       GADGET igadget19,150,125,4,border,0,strinfo18,18
000020AE igadget19       GADGET 0,80,150,1,border,string6,0,111

000020DA strinfo11       STRINF expx1
000020FE strinfo12       STRINF expx1+16*1
00002122 strinfo13       STRINF expx1+16*2
00002146 strinfo14       STRINF expx1+16*3
0000216A strinfo15       STRINF expx1+16*4
0000218E strinfo16       STRINF expx1+16*5
000021B2 strinfo17       STRINF expx1+16*6
000021D6 strinfo18       STRINF expx1+16*7
```

```
000021FA expx1          ZAHLENSTRINGS "0","1","2","3","4","5","6","7"

         * Parameter

0000227A igadget20      GADGET igadget21,140,30,4,border,string2,strinfo20,20
000022A6 igadget21      GADGET igadget22,140,60,4,border,string3,strinfo21,21
000022D2 igadget22      GADGET igadget23,140,90,4,border,string4,strinfo22,22
000022FE igadget23      GADGET igadget24,140,120,3,regler,string5,propinfo,23
0000232A igadget24      GADGET 0,80,150,1,border,string6,0,111

00002356 strinfo20        STRINF Untere
0000237A strinfo21        STRINF Obere
0000239E strinfo22        STRINF Schritt

         * Ausgabefenster

000023C2 igadget30        GADGET igadget31,10,140,1,border,string8,0,112
000023EE igadget31        GADGET igadget32,185,140,1,border,string9,0,113
0000241A igadget32        GADGET 0,105,160,1,border,string10,0,111

00002446 Untere           dc.b   "-2",0,0,0,0,0,0,0,0,0,0,0,0,0
                          even
00002456 Obere            dc.b   "2",0,0,0,0,0,0,0,0,0,0,0,0,0,0
                          even
00002466 Schritt          dc.b   "0.5",0,0,0,0,0,0,0,0,0,0,0,0
                          even

00002476 Unterex1         dc.b   "-2",0,0,0,0,0,0,0,0,0,0,0,0,0
                          even
00002486                  dc.b   "2",0,0,0,0,0,0,0,0,0,0,0,0,0,0
                          even
00002496                  dc.b   "0.5",0,0,0,0,0,0,0,0,0,0,0,0
                          even


000024A6 fakzaehler     ZAHLENSTRINGS "1","1","1","1","1","1","1","1"

00002526 expzaehler     ZAHLENSTRINGS "0","1","2","3","4","5","6","7"

000025A6 faknenner      ZAHLENSTRINGS "1","1","1","1","1","1","1","1"

00002626 expnenner      ZAHLENSTRINGS "0","1","2","3","4","5","6","7"

000026A6 floatfakz      ZAHLENFLOAT $80,$41,$80,$80,$80,$41,$41

000026C6 floatexpz      ZAHLENFLOAT 0,0,$C0,$A0,$E0,$42,$43
```

```
000026E6 floatfakn     ZAHLENFLOAT $80,$41,$80,$80,$80,$41,$41

00002706 floatexpn     ZAHLENFLOAT 0,0,$C0,$A0,$E0,$42,$43

         * Fließkommazahlen

00002726 floatuntere    dc.b    $80,0,0,$C2

0000272A floatobere     dc.b    $80,0,0,$42

0000272E floatschritt   dc.b    $80,0,0,$40

00002732 xwert          ds.l  1000

000036D2 ywert          ds.l  1000

00004672 zaehlwert      ds.l  1000

00005612 nennwert       ds.l  1000
```

Mehr als die Hälfte des Objectcodes unseres Programms besteht aus Strukturen. An dieses Bild wird man sich gewöhnen müssen, wenn man benutzerfreundliche Programme entwickeln möchte. Man sieht auch, daß es sich ausgezahlt hat, eingangs relativ viele Makros zu definieren, da man auf diese Weise unglaublich viel Schreibarbeit spart. Stellen Sie sich vor, wir hätten alle Gadget-Strukturen ausführlich hinschreiben müssen! Nein, wir können wirklich glücklich sein, daß uns der DEVPAC-Assembler soviel Arbeit abnehmen kann.

Falls Sie die Startstrings der Stringgadgets ändern möchten, vergessen Sie nicht, auch die entsprechenden FFP-Zahlen zu ändern, da prinzipiell nur diese für das Programm relevant sind. Differenzen zwischen einer FFP-Zahl und deren String können zu merkwürdigen Ergebnissen führen, wenn auch die Gefahr einer Guru-Meditation im allgemeinen nicht besteht.

Zum Schluß noch ein Wort an die glücklichen Besitzer einer RAM-Erweiterung: Wie Sie sicherlich wissen, müssen alle Grafik-Daten von Images in den unteren 512 Kbyte des Amigas (Chip-RAM) liegen, da sonst der Grafikchip keinen Zugriff auf diese Bits hat. Wenn Sie Pech haben, liegt der Wertetabelle-Code nach dem Starten in der anderen Speicherhälfte. Man erkennt dies daran, daß der Pfeil des Proportionalgadgets nicht daran denkt, auf dem Bildschirm zu erscheinen. Nun könnte ich Ihnen raten, die Erweiterung softwaremäßig abzuschalten, wie dies andere Autoren zu tun pflegen. Die elegantere Lösung besteht aber darin, das Gadget einfach in den Typ AUTOKNOB umzuwandeln, da Intuition dann selbst für die Grafikdaten zuständig ist. Es braucht wohl nicht gesondert erwähnt zu werden, daß unser Betriebssystem so »intelligent« ist, sich die Grafikdaten in den Speicherbereich zu legen, in dem es auch auf sie zugreifen kann! Wenn Sie also beim Label »propinfo« das erste Wort (Adresse $1B78) in den Wert 3 ändern, setzen Sie das Bit 0 und damit die AUTOKNOB-Funktion. Sie werden nun auch mit RAM-Erweiterung einen Regler sehen können.

Grafikprogrammierung auf dem Amiga

Kapitel 4

Wenn man nach der herausragenden Eigenschaft des Amiga fragt, wird man in 99,9 % aller Fälle die grafischen Möglichkeiten als Antwort genannt bekommen. In der Tat kann man lange einen Rechner suchen, der in einer so großen Auflösung mit soviel Farben so schnell zeichnen kann, wie es uns ja immer wieder von Intuition vor Augen geführt wird. Im Gegensatz zu Computern wie dem C64 braucht man deshalb überhaupt keinen Textmodus einzuführen, man bewegt sich ständig in der hochauflösenden Grafik, wobei der Text nichts anderes ist als eine sinnvolle Ansammlung von Grafikpunkten. Dieses Verfahren ist nur dann anzuwenden, wenn mit sehr hoher Geschwindigkeit gezeichnet werden kann.

Selbstverständlich wird die Grafik hinreichend vom DOS unterstützt. Hierzu kann man die Graphics-Library verwenden, die eine Menge an Befehlen enthält, von der man besonders als Umsteiger von einem Kleincomputer zunächst erschlagen wird. In diesem Buch können wir natürlich nicht auf alle Funktionen eingehen, werden aber die Mehrzahl, insbesondere die wichtigen, besprechen. Dazu kann man sie in drei Gruppen einteilen:

a) **die »normalen« Grafikbefehle**
Zu dieser Gruppe wollen wir die Befehle zählen, die sich mit der Farbgebung, der Mustererstellung, der Behandlung von Bildschirmausschnitten (z.B. Scrolling) und dem Zeichnen beliebiger Figuren vom Punkt bis hin zur Ellipse beschäftigen.

b) **die »Area«-Befehle**
Einen speziellen Bonbon stellen diejenigen Befehle dar, die nicht rein softwaremäßig auf die Bitplanes zugreifen, indem bestimmte Bits gesetzt und gelöscht werden, sondern den sogenannten Blitter zur Grafikerstellung heranziehen. Hierbei handelt es sich um ein Hardwareteil, welches in extrem kurzer Zeit riesige Datenmengen verschieben kann. Eine genaue Beschreibung des Blitters würde den Rahmen dieses Buches sprengen, ich möchte hierfür auf das Buch »Amiga Intern« verweisen.
Wichtig für uns ist ausschließlich die Wirkung des Blitters, da er ja von Library-Funktionen angesteuert wird. Diese besteht darin, daß ausgefüllte Kreise und Polygone mit extrem hoher Geschwindigkeit gezeichnet werden können. Dazu ist aber etwas mehr Programmieraufwand erforderlich, da u.a. Speicherplatz für die Blitteroperationen bereitgestellt werden muß.

c) **Die Zeichensatz-Befehle**
Wenn Sie sich einmal das Direktory Ihrer Workbench-Diskette ansehen, werden Sie feststellen, daß sich dort auch mehrere Zeichensätze verborgen halten, die man u.a. im Programm »Notepad« verwenden kann. Leider sind bis heute, genausowenig wie zu den Area-Befehlen, irgendwelche Informationen zu bekommen, wie man auf diese Zeichensätze von Assembler aus zugreifen kann. Ich habe mich daher besonders intensiv mit diesem Thema auseinandergesetzt, so daß Sie, liebe Leser, in Zukunft Ihre eigenen Programme mit Fremdzeichensätzen nur so füttern können.

Leider ist aber, wie immer bei Commodore, ein Haar in der Suppe zu finden. So kann man mit der Ellipsenroutine nur ganze Ellipsen oder Kreise zeichnen, der häufig auftretende Fall von Ellipsenausschnitten kann nicht behandelt werden. Deshalb habe ich nach unserem Beispielprogramm, einer Chart-Grafik mit 3-D-Tortendiagramm und 3-D-Säulengrafik, welches gerade dieses Defizit aufzeigt, eine eigene Ellipsenroutine entworfen. Wenn man schon einmal dabei ist, etwas zu verbessern, so können wir uns auch gleich der Zeichengeschwindigkeit zuwenden. Durch direkten Zugriff auf die Bitplanes wird unsere Ellipsenroutine in der bislang auf einem Homecomputer noch nicht erreichten Geschwindigkeit von 12 600 Punkten pro Sekunde zeichnen können. Die Geschwindigkeit steht völlig unabhängig von Größe und Aussehen der Ellipse zur Verfügung.

Eine weitere Unzulänglichkeit ist darin zu sehen, daß für einige Operationen wie z.B. die Mustererstellung keine Library-Funktionen existieren. Wir müssen deshalb eigene Makros entwickeln, die am besten in das Grafik-Include-File integriert werden, wie es in Kapitel 1 dargelegt wurde.

Schließlich wird auch noch auf das IFF-Bildformat eingegangen. Da es endlich gelungen ist, ein Standard-Format für Grafiken zu finden, sollte man sich auch in den eigenen Programmen daran halten, um Bilder von Fremdprogrammen übernehmen zu können und umgedreht. Im folgenden Kapitel wird hierzu auch ein Beispielprogramm behandelt, in welchem man Screens im IFF-Format auf Disketten speichern und wieder laden kann. Später wird im Kapitel »Devices« auch gezeigt, wie man Grafiken ausdrucken kann.

Zunächst wollen wir es aber langsam angehen lassen und uns mit den »normalen« Grafikfunktionen beschäftigen. Um die Makros benutzen zu können, muß das Include-File »graphics_lib.i« geladen werden, wie es in Kapitel 1 beschrieben wurde.

4.1 Die »normalen« Grafikfunktionen

Alle Grafikfunktionen haben eins gemeinsam: sie benötigen den Rastport des aktuellen Windows, in das gezeichnet werden soll. Da oben schon oft beschrieben wurde, wie man an den Rastport herankommt, erlauben Sie mir, um Platz zu sparen, die Abkürzung RP für den Zeiger auf die Rastport-Struktur, den Sie aus der Window-Struktur gewinnen können. Einige Befehle wirken aber immer auf den gesamten Screen, wie z.B. die Farbgebung eines Registers. Hierfür reicht der Rastport nicht mehr aus. Man benötigt an dieser Stelle den

View-Port des Screens. Wie wir aus Kapitel 2 wissen, beginnt die View-Port-Struktur an der Adresse $2C in der Screen-Struktur:

```
move.l screenpointer,d0
add.l #$2C,d0
move d0,VP
```

Der benötigte Zeiger, den ich im folgenden mit VP abkürzen werde, steht damit im Register D0 zur Verfügung.

4.1.1 Die Farbgestaltung

a) Farbregister setzen: **SetRGB4**
Diese Funktion dient dazu, ein Farbregister mit einem spezifizierten Farbwert zu laden. Dazu müssen die Intensitäten der Grundfarben Rot, Grün und Blau angegeben werden, die sich jeweils von 0 bis 15 bewegen dürfen.

```
move.l VP,a0                ;Zeiger auf View-Port des Screens
move.l #register,d0         ;Farbregisternummer
move.l #rot,d1              ;Farbanteil Rot (0-15)
move.l #grün,d2             ;Farbanteil Grün (0-15)
move.l #blau,d3             ;Farbanteil Blau (0-15)
CALLGRAF SetRGB4            ;Makro aufrufen
```

b) Farbregister auswerten: **GetRGB4**
Für manche Anwendungen ist es erforderlich, die Farbanteile der einzelnen Register auszulesen, z.B. für das Speichern einer Grafik im IFF-Format (Kapitel 5). Die GetRGB4-Funktion liefert ein Wort, in dem für jede Farbe 4 Bit reserviert sind:

Bits 12–15:	Unbenutzt
Bits 8–11:	Farbanteil Rot
Bits 4– 7:	Farbanteil Grün
Bits 0– 3:	Farbanteil Blau

Folgende Routine analysiert die Farbanteile heraus, wobei ein Zeiger auf die Color-Map übergeben werden muß. Dieser steht in einem Langwort in der View-Port-Struktur, so daß es für uns kein Problem ist, an ihn heranzukommen:

```
move.l VP,a0                ;Zeiger auf View-Port des Screens
move.l 4(a0),a0             ;Zeiger auf Color-Map
move.l #register,d0         ;Farbregisternummer
CALLGRAF GetRGB4            ;Makro aufrufen
move.w d0,d1                ;Wert sichern
and.w #%0000111100000000,d0 ;Bits für Rot isolieren
lsr.w #8,d0                 ;um 8 Bit nach rechts verschieben
move.b d0,rot               ;Farbanteil Rot (0-16)
move.b d1,d0                ;Bits für Blau und Grün zurückkopieren
and.b #%11110000,d0         ;Bits für Grün isolieren
lsr.b #4,d0                 ;um 4 Bit nach rechts verschieben
move.b d0,grün              ;Farbanteil Grün (0-16)
```

```
and.b #%00001111,d1        ;Bits für Blau isolieren
move.b d1,blau             ;Farbanteil Blau (0-16)

rot   dc.b 0
grün  dc.b 0
blau  dc.b 0
```

c) Zeichenfarbe setzen: **SetAPen**
Hiermit legen Sie die Zeichenfarbe für alle »normalen« Grafikbefehle wie Punkt setzen, Linie ziehen etc. fest. Dafür wird die Nummer eines Farbregisters ausgewählt. Falls Ihnen keine aktuelle Farbe zusagt, können Sie die eines Registers mit SetRGB4 nach Ihren Vorstellungen anpassen.

```
move.l RP,a1               ;Zeiger auf Rastport des Windows
move.l #register,d0        ;Farbregisternummer
CALLGRAF SetAPen           ;Makro aufrufen
```

d) Hintergrundfarbe setzen: **SetBPen**
Diese Funktion arbeitet analog zu SetAPen, wobei jedoch die Hintergrundfarbe des Windows bestimmt wird.

```
move.l RP,a1               ;Zeiger auf Rastport des Windows
move.l #register,d0        ;Farbregisternummer
CALLGRAF SetBPen           ;Makro aufrufen
```

4.1.2 Die Zeichenbefehle

a) Grafikcursor setzen: **Move**
Die Einrichtung eines Grafikcursors erleichtert besonders das Zeichnen von Polygonen. Es handelt sich praktisch um einen unsichtbaren Koordinatenspeicher, der den Startpunkt für eine zu ziehende Linie angibt. Da die Koordinaten des Endpunktes der Linie automatisch wieder in den Grafikcursor gesetzt werden, kann man von einem Punkt ausgehend viele Linien hintereinander zeichnen, wobei immer nur der folgende Punkt behandelt zu werden braucht. Nach dem Draw-Befehl werde ich dies an Hand eines Beispiels erläutern.

```
move.l RP,a1               ;Zeiger auf Rastport des Windows
move.l #xkoord,d0          ;x-Koordinate des Grafikpunktes
move.l #ykoord,d1          ;y-Koordinate des Grafikpunktes
CALLGRAF Move              ;Grafikcursor setzen
```

b) Linie vom Grafikcursor zu einem Punkt ziehen: **Draw**
Bei der Angabe von nur einem Koordinatenpaar wird eine Linie vom Grafikcursor zu diesem Punkt gezogen. Es wird in der Farbe gezeichnet, die durch SetAPen festgelegt wurde.

```
move.l RP,a1                ;Zeiger auf Rastport des Windows
move.l #xkoord,d0           ;x-Koordinate des Grafikpunktes
move.l #ykoord,d1           ;y-Koordinate des Grafikpunktes
CALLGRAF Draw               ;Linie ziehen
```

Um z.B. ein Quadrat mit den Eckpunkten (10,10), (10,40), (40,40) und (40,10) zu zeichnen, wäre folgende Befehlssequenz erforderlich:

```
move.l RP,a1                ;Zeiger auf Rastport des Windows
move.l #10,d0               ;x-Koordinate linke obere Ecke
move.l #10,d1               ;y-Koordinate linke obere Ecke
CALLGRAF Move               ;Grafikcursor setzen
move.l #10,d0               ;x-Koordinate linke untere Ecke
move.l #40,d1               ;y-Koordinate linke untere Ecke
CALLGRAF Draw               ;Linie ziehen
move.l #40,d0               ;x-Koordinate rechte untere Ecke
move.l #40,d1               ;y-Koordinate rechte untere Ecke
CALLGRAF Draw               ;Linie ziehen
move.l #10,d0               ;x-Koordinate rechte obere Ecke
move.l #40,d1               ;y-Koordinate rechte obere Ecke
CALLGRAF Draw               ;Linie ziehen
move.l #10,d0               ;x-Koordinate Ausgangspunkt
move.l #10,d1               ;y-Koordinate Ausgangspunkt
CALLGRAF Draw               ;Linie ziehen
```

c) Punkt setzen: **WritePixel**
Im Gegensatz zu Move wird hier unmittelbar ein Grafikpunkt gesetzt. Da leider keine Funktion zum Löschen eines Punktes vorhanden ist, muß man sich damit behelfen, vor Ausführung dieser Funktion die Zeichen- auf die Hintergrundfarbe und anschließend wieder auf die Original-Zeichenfarbe zu setzen.

```
move.l RP,a1                ;Zeiger auf Rastport des Windows
move.l #xkoord,d0           ;x-Koordinate des Grafikpunktes
move.l #ykoord,d1           ;y-Koordinate des Grafikpunktes
CALLGRAF WritePixel         ;Punkt setzen
```

d) Kreis oder Ellipse zeichnen: **DrawEllipse**
Wie schon oben erwähnt, kann man ausschließlich komplette Ellipsen zeichnen (360 Grad). Eine spezielle Kreisfunktion existiert nicht, wenn dies auch z.B. im Programmierhandbuch auf Seite 98 behauptet wird. Zwar kann ein C-Compiler etwas mit dem Ausdruck DrawCircle anfangen, intern ruft er aber die DrawEllipse-Routine auf, nachdem er x- und y-Radius gleichgesetzt hat. Daher können Sie bei einem Kreis auch keine höhere Zeichengeschwindigkeit erwarten. Die Funktion arbeitet so langsam, daß man bei großen Ellipsen den Aufbau mit dem Auge mitverfolgen kann.

```
move.l RP,a1                ;Zeiger auf Rastport des Windows
move.l #xmittel,d0          ;x-Koordinate des Kreis/Ellipsenmittelpunktes
move.l #ymittel,d1          ;y-Koordinate des Kreis/Ellipsenmittelpunktes
move.l #xradius,d2          ;Radius in x-Richtung
move.l #yradius,d3          ;Radius in y-Richtung
CALLGRAF DrawEllipse        ;Kreis/Ellipse zeichnen
```

Achtung: Bevor Sie dieses Makro aufrufen können, müssen Sie in älteren DEVPAC-Versionen das Include-File korrigiert haben, wie dies in Kapitel 1 beschrieben wurde!

e) Ein Polygon zeichnen: **PolyDraw**
Mit Hilfe dieser Funktion ist es wesentlich schneller möglich, ein Polygon zu zeichnen als mit den Befehlen Move und Draw. Allerdings müssen schon vor Zeichenbeginn die Zahl der Eckpunkte und deren Koordinaten bekannt sein. Werden diese erst nach und nach berechnet, muß man mit den Grafikcursor-Funktionen vorlieb nehmen.

```
move.l RP,a1             ;Zeiger auf Rastport des Windows
move.l #anzahl,d0        ;Anzahl der Koordinatenpaare
move.l #koordtab,a0      ;Zeiger auf Koordinatentabelle
CALLGRAF PolyDraw        ;Polygon zeichnen

koordtab  dc.w x1,y1,x2,y2,x3,y3,...,xanzahl,yanzahl
```

f) Ein ausgefülltes Rechteck zeichnen: **RectFill**
Mit dieser Funktion kann man nicht nur Rechtecke zeichnen, sondern auch elegant die fehlende Funktion zum Löschen eines beliebigen Windowbereiches ersetzen: Dazu wird, wie im Wertetabellen-Programm gezeigt, vor dem Aufruf der Funktion die Zeichenfarbe auf den Wert der Hintergrundfarbe umgestellt. Man darf nur nicht vergessen, hinterher die alte Zeichenfarbe wiederherzustellen, sonst kann man lange, lange auf weitere Zeichnungen warten...

```
move.l RP,a1             ;Zeiger auf Rastport des Windows
move.l #xlo,d0           ;x-Koordinate der linken oberen Ecke
move.l #ylo,d1           ;y-Koordinate der linken oberen Ecke
move.l #xru,d2           ;x-Koordinate der rechten unteren Ecke
move.l #yru,d3           ;y-Koordinate der rechten unteren Ecke
CALLGRAF RectFill        ;ausgefülltes Rechteck zeichnen
```

4.2 Die Area-Befehle

Wie oben schon erwähnt, arbeiten diese Funktionen mit dem Hardware-Blitter. Um die Areas für die Benutzung durch den Blitter zu installieren, ist folgender prinzipieller Ablauf erforderlich:

a) Reservierung von Speicherplatz
b) Installierung des Speicherbereiches für die Nutzung durch Areas
c) Ausführen der Area-Funktionen
d) Freigeben des Speicherplatztes

4.2.1 Die Bereitstellung von Speicherplatz : AllocRaster

Bevor man irgend etwas initialisieren kann, muß man erst einmal den benötigten Speicherplatz vom System reserviert bekommen. Dazu muß man der AllocRaster-Funktion die Größe des Bildschirmausschnitts mitteilen, der von der größten zu zeichnenden Figur benötigt wird. Da dieser aber nicht immer bekannt ist, sollte man es sich angewöhnen, immer die Koordinaten des gesamten Screens zu übergeben. Dies geschieht in Pixeln jeweils für Höhe und Breite, z.B. beim HIRES-Modus:

```
move.l #640,d0        ;Breite in Pixeln
move.l #256,d1        ;Höhe in Pixeln
CALLGRAF AllocRaster  ;Speicher reservieren
move.l d0,mempointer  ;Adresse des Speicherbeginns merken
mempoint  dc.l 0
```

Der Pointer darf auf gar keinen Fall verloren gehen, da man ihn u.a. zum Freigeben des Speichers benötigt.

4.2.2 Speicher für Verwendung durch Areas initialisieren: InitTmpRas

Damit die Area-Befehle nutzbar werden, muß eine Struktur mit dem Namen »TmpRas« angelegt werden, in der die Speicheradresse sowie die Größe in Bytes abgelegt werden. Man muß dafür in seinem Programm zwei Langwörter bereitstellen und einen Zeiger auf diesen Bereich übergeben. Die Größe des mit AllocRaster reservierten Speichers erhält man, indem man (Breite * Höhe)/8 rechnet, wie dies in Kapitel 1 im Abschnitt über die Screens auch schon beschrieben wurde. In unserem Fall wäre damit der Wert 40 960 herzunehmen:

```
move.l mempoint,a1     ;Zeiger auf reservierten Speicher
move.l #40960,d0       ;Größe in Bytes
move.l #tmpras,d1      ;Zeiger auf TmpRas-Struktur
CALLGRAF InitTmpRas    ;Speicher initialisieren
move.l RP,a0           ;Zeiger auf Rastport
move.l #tmpras,$0C(a0) ;Zeiger auf TmpRas-Struktur in Rastport
                       ;eintragen

tmpras   dc.l 0,0
```

4.2.3 Die Area-Grafikfunktionen

a) Ausfüllen einer Fläche: **Flood**
Nun können wir schon unseren ersten Area-Befehl anwenden. Er ermöglicht es, beliebige Flächen mit diversen Mustern zu füllen. Wie man die Muster erstellen kann, sehen wir im übernächsten Abschnitt. Neben dem Rastport-Zeiger und den Startkoordinaten des Punktes, von dem an gefüllt werden soll, muß auch der sogenannte

Füllmodus übergeben werden: Hierbei kann man wählen zwischen einer Füllung des Bereiches, der die gleiche Farbe hat wie der Startpunkt (1) oder des Bereiches, der durch die Farbe eingegrenzt wird, die durch die Funktion SetOPen (siehe unten) festgelegt wurde (0). Bei letzterem Modus wird also keine Rücksicht darauf genommen, ob irgendwelche Figuren, die mit der Zeichenfarbe (SetAPen) gemalt wurden, überschrieben werden. Im Normalfall wird man den ersten Modus verwenden.

```
move.l RP,a1              ;Zeiger auf Rastport des Windows
move.l #xstart,d0         ;x-Koordinate des Startpunktes
move.l #ystart,d1         ;y-Koordinate des Startpunkts
move.l #mode,d2           ;Zeichenmode (0 oder 1)
CALLGRAF Flood            ;Fläche ausfüllen
```

b) Area-Liste initialisieren: **InitArea**
Bevor man die weiteren Funktionen wie Zeichnen von ausgefüllten Kreisen und Polygonen nutzen kann, muß man noch eine weitere Struktur mit dem Namen »areaInfo« anlegen. Man muß wissen, daß Area-Figuren nicht schlagartig nach dem eigentlichen Zeichenbefehl gezeichnet, sondern nur die Koordinaten in eine spezielle Liste eingetragen werden. Wenn man alle Figuren »virtuell« gezeichnet hat, gibt es einen weiteren Befehl, durch den dann alle in die Liste eingetragenen Figuren blitzschnell auf den Bildschirm gebracht werden. Deshalb ist es notwendig, Platz für die Koordinaten zu reservieren. Ebenfalls übergeben werden muß natürlich auch die maximale Anzahl der Koordinateneinträge. Da pro Eintrag 5 Byte benötigt werden, kann man sich den Speicherbedarf sofort ausrechnen. In die AreaInfo-Struktur werden die Start- und Endadresse dieses Koordinatenspeichers sowie die Zahl der maximalen Einträge vermerkt. Für sie müssen vier Langworte reserviert werden:

```
move.l #areainfo,a0       ;Zeiger auf AreaInfo-Struktur
move.l #platz,a1          ;Zeiger auf Speicher für Koordinateneinträge
move.l #zahl,d0           ;Max. Zahl der Koordinateneinträge
CALLGRAF InitArea         ;AreaInfo-Struktur initialisieren

areainfo    ds.l 4
platz       ds.w 5/2 * zahl
```

Der Pointer auf die Area-Info-Struktur muß unbedingt in den Rastport eingetragen werden, da alle weiteren Befehle sonst ohne Wirkung bleiben. Hierfür ist das Langwort ab der Adresse $10 zuständig:

```
move.l #areainfo,$10(RP)
```

Bitte beachten Sie, daß bei einer zu geringen Größe von »platz« die weiteren Daten, wahrscheinlich also Ihr Programmcode, mit Koordinaten überschrieben werden. Damit kann man aber fast schon seinen Amiga auf ein Erscheinen des Gurus verwetten! Für den Flood-Befehl ist diese Initialisierung nicht erforderlich.

c) Startpunkt für Polygon definieren: **AreaMove**
Durch die Area-Move-Funktion wird der Startpunkt für ein Polygon definiert, ähnlich wie es durch das Setzen des Grafikcursors bei den »normalen« Befehlen geschehen ist. Gleichzeitig wird eine etwaige Definition eines vorherigen Polygons beendet.

```
move.l RP,a1            ;Zeiger auf Rastport
move.l #xkoord,d0       ;x-Koordinate
move.l #ykoord,d1       ;y-Koordinate
CALLGRAF AreaMove       ;Koordinaten in Liste eintragen
```

Ergebnis in D0: 1 (Fehler aufgetreten) oder 0 (alles O.K.)

d) Eckpunkt für Polygon definieren: **AreaDraw**
Mit dieser Funktion werden die Koordinaten der Eckpunkte des Polygons übergeben. Zum Schließen des Polygons ist es erforderlich, am Schluß nochmals die AreaMove-Funktion aufzurufen, die intern das Polygon als abgeschlossen definiert. Durch die Übergabe eines weiteren Koordinatenpaares kann man gleichzeitig den Startpunkt eines weiteren Polygons festlegen.

```
move.l RP,a1            ;Zeiger auf Rastport
move.l #xkoord,d0       ;x-Koordinate
move.l #ykoord,d1       ;y-Koordinate
CALLGRAF AreaDraw       ;Koordinaten in Liste eintragen
```

Ergebnis in D0: 1 (Fehler aufgetreten) oder 0 (alles O.K.)

e) Kreis oder Ellipse definieren: **AreaEllipse**
Neben den Polygonen können Sie auch ausgefüllte Ellipsen mit einer sehr hohen Geschwindigkeit zeichnen lassen. Dazu werden die Koordinaten vom Mittelpunkt und der Radien in die Liste eingetragen. Eine AreaCircle-Funktion existiert nicht, diesbezügliche Angaben wie z.B. im Programmierhandbuch beziehen sich nur intern auf die C-Compiler, die dann einfach x- und y-Radius gleichsetzen.
Bitte beachten sie wie schon bei der DrawEllipse-Funktion, daß diese in älteren DEVPAC-Versionen nicht implementiert wurde. Die Modifikation des Include-Files ist in Kapitel 1 nachzulesen.

```
move.l RP,a1           ;Zeiger auf Rastport
move.l #xmittel,d0     ;x-Koordinate des Kreis/Ellipsenmittelpunktes
move.l #ymittel,d1     ;y-Koordinate des Kreis/Ellipsenmittelpunktes
move.l #xradius,d2     ;x-Radius
move.l #yradius,d3     ;y-Radius
CALLGRAF AreaEllipse   ;Koordinaten in Liste eintragen
```

Ergebnis in D0: 1 (Fehler aufgetreten) oder 0 (alles O.K.)

f) Zeichnen aller Area-Figuren: **AreaEnd**
Durch diese Funktion werden schließlich alle Figuren, die in die Liste eingetragen wurden, auf den Bildschirm gezeichnet. Die Zeichengeschwindigkeit ist so hoch, daß man den Aufbau einzelner Figuren praktisch nicht nachvollziehen kann. Man sollte

sich jedoch hüten, diesen Befehl aufzurufen, falls bei den Eintragungen etwas schiefgelaufen sein sollte. Ein Fehler deutet dort nämlich fast immer darauf hin, daß nicht genügend Speicherplatz reserviert wurde. Deshalb könnten einige Figuren unvollständig eingetragen sein, was zumindest merkwürdige Grafiken, eventuell auch den Guru hervorrufen würde.

```
move.l RP,a1                ;Zeiger auf Rastport
CALLGRAF AreaEnd            ;Figuren zeichnen
```

Die Area-Liste wird vollständig gelöscht, so daß sie neu benutzt werden kann, ohne irgendwelche Pointer neu zu setzen.

4.2.4 Speicherplatz freigeben: FreeRaster

Nach Beendigung der Arbeit mit den Area-Befehlen ist es eminent wichtig, den mit AllocRaster reservierten Speicherplatz wieder freizugeben. Dazu dient die Funktion FreeRaster, die wir mit dem Speicherpointer sowie erneut den Ausmaßen des Screen(bereiche)s versorgen müssen. Die Ausmaße müssen unbedingt mit denen der Alloc-Funktion übereinstimmen, sonst kann sich der Guru schon einmal warmlaufen.

```
move.l mempointer,a0        ;Speicherpointer holen
move.l #640,d0              ;Breite in Pixeln
move.l #256,d1              ;Höhe in Pixeln
CALLGRAF AllocRaster        ;Speicher freigeben
```

4.3 Sonstige Grafikfunktionen

a) Zeichenmode setzen: **SetDrMd**
Bislang sind wir stets davon ausgegangen, daß mit der Farbe gezeichnet wird, die mit der Funktion SetAPen festgelegt wurde. Dieses ist aber nur einer von vier möglichen Zeichenmodi (JAM1). Daneben existiert noch der soganannte JAM2-Modus, der für Füllmuster (z.B. RectFill) verwendet wird. Dabei wird bei gesetztem Bit die Vordergrundfarbe genommen, während bei gelöschtem Bit die durch SetBPen festgelegte Hintergrundfarbe zur Anwendung kommt. Beide Zeichenmodi können auch komplementiert werden. Für die Textdarstellung existiert noch ein invertierender Modus für jeweils JAM1 und JAM2. Somit stehen sechs Modi zur Verfügung, deren Zusammenhang mit dem an die SetDrMd-Funktion zu übergebenen Wert folgende Tabelle zeigt:

Zeichenmodus	Wert
JAM1	0
JAM2	1
JAM1 komplementiert	2

Zeichenmodus	Wert
JAM2 komplementiert	3
JAM1 invertiert	4
JAM2 invertiert	5

```
move.l RP,a1              ;Zeiger auf Rastport
move.l #wert,d0           ;Zeichenmodus
CALLGRAF SetDrMd          ;setzen
```

b) Window einfärben : **SetRast**
Mit dieser Funktion ist es möglich, ein gesamtes Window einzufärben. Allerdings gehen dabei auch die Titelleiste sowie alle Systemgadgets verloren.

```
move.l RP,a1              ;Zeiger auf Rastport
move.l #farbe,d0          ;Nummer des Farbregisters
CALLGRAF SetDraw
```

c) Windowbereich scrollen: **ScrollRaster**
Mit Hilfe dieser Funktion wird es ermöglicht, einen beliebigen Windowausschnitt relativ schnell zu verschieben, wobei der freigewordene Bereich mit der Hintergrundfarbe (SetBPen) gefüllt wird. Neben dem oberen linken und unteren rechten Eckpunkt des zu scrollenden Rechtecks muß auch der Offset in Pixeln angegeben werden, um den verschoben werden soll. Negative Werte bedeuten dabei, daß nach unten bzw. rechts gescrollt wird, positive dementsprechend eine Verschiebung nach links bzw. oben.

```
move.l RP,a1              ;Zeiger auf Rastport
move.l #dx,d0             ;Offset in x-Richtung
move.l #dy,d1             ;Offset in y-Richtung
move.l #xlo,d2            ;x-Koordinate der linken oberen Ecke
move.l #ylo,d3            ;y-Koordinate der linken oberen Ecke
move.l #xru,d4            ;x-Koordinate der rechten unteren Ecke
move.l #xru,d5            ;y-Koordinate der rechten unteren Ecke
CALLGRAF ScrollRaster     ;Window-Ausschnitt verschieben
```

4.4 Makros zur Musterdefinition

In den letzten Abschnitten haben wir einige Befehle kennengelernt, die eine Fläche ausfüllen. Nun kann durchaus der Wunsch auftreten, daß die Flächen doch bitte nicht immer ganz, sondern in Mustern ausgefüllt werden mögen. Ebenso wäre es schön, wenn man Linien nicht immer ganz durchgezogen zeichnen müßte, sondern z.B. gestrichelt darstellen könnte.

An dieser Stelle versagt die Graphics-Library völlig, da keine Funktion existiert, die eine Editierung der Muster zuließe (zumindest nicht in der Kickstart-Version 1.2, die diesem Buch zugrunde liegt).

Prinzipiell bietet Intuition aber die Möglichkeit, eigene Muster zu definieren. In Kapitel 1 habe ich Ihnen daher zwei Makros vorgestellt, die in das Graphics-Include File integriert wurden. Sie können damit so auf sie zugreifen, als seien sie Bestandteil der Graphics-Library.

a) Musterdefinition für Linien: **SetDrPt**
Mit Hilfe dieses Makros ist es möglich, ein Muster für alle Funktionen zu definieren, die Linien ziehen (Draw, AreaDraw). Dabei wird das Muster in einer Wortvariablen gespeichert, wobei ein gesetztes Bit einem gesetzten Punkt entspricht. Nach maximal 16 Grafikpunkten wiederholt sich also das Muster. Hierbei handelt es sich nicht um eine Gemeinheit von mir, sondern um eine durch den reservierten Platz in der Rastport-Struktur festgelegte Tatsache.

```
SetDrPt         macro
                move.w d0,$22(a1)
                move.w $20(a1),d0
                or.w #1,d0
                move.w d0,$20(a1)
                move.b #15,$1F(a1)
                endm
```

Damit werden das Muster sowie ein Flag und die Anzahl der zu betrachtenden Punkte pro Muster in die Rast-Port-Struktur eingetragen. Übergeben werden der Rastport-Zeiger sowie das Wort mit der Musterdefinition.

```
move.l RP,a1                    ;Zeiger auf Rastport
move.w muster,d0                ;Musterdefinition
SetDrPt                         ;Funktion aufrufen

muster dc.w %1010101010101010  (punktierte Linie)
```

b) Musterdefinition für Flächen : **SetAfPt**
Die Musterdefinition funktioniert prinzipiell wie bei SetDrPt. Allerdings kann man hier nicht nur ein Wort, sondern beliebig viele angeben, weil man es ja mit einer zweidimensionalen Fläche zu tun hat. Das erste Wort definiert dabei das Muster für die oberste Zeile der Fläche, das zweite das Muster für die darunterliegende Zeile usw. Während also in x-Richtung nach spätestens 16 Punkten eine Wiederholung des Musters auftritt, kann man in y-Richtung theoretisch beliebig viele Muster definieren, so daß sich der Inhalt einer Zeile nicht undedingt wiederholen muß. Allerdings kann man die Anzahl der in y-Richtung darstellbaren Muster nur in Zweierpotenzen festlegen, d.h., es müssen 1,2,4,8,16,... Worte für die Definition verwendet werden.

```
SetAfPt         macro
                move.l d0,$08(a1)
                move.b d1,$1D(a1)
                endm
```

Übergeben werden neben dem obligatorischen Zeiger auf den Rastport ein weiterer Zeiger auf das Füllmuster sowie die Zahl der Wörter, indem der Exponent der Zweierpotenz verwendet wird (z.B. 3 entspricht 23=8 Wörtern)

```
move.l RP,a1                    ;Zeiger auf Rastport
move.l #füll,d0                 ;Zeiger auf Füllmuster
move.l #2,d1                    ;Zahl der Muster-Wörter (hier 22=4)
SetAfPt                         ;Funktion aufrufen
```

```
füll  dc.w %1111111111111111  (oberste Zeile ganz zeichnen)
      dc.w %1100110011001100  (darunterliegende Zeile strichpunktiert)
      dc.w %1010101010101010  (darunterliegende Zeile punktiert)
      dc.w %0000000000000000  (darunterliegende Zeile gar nicht)
```

Von der fünften Zeile an wiederholt sich das Muster auch in y-Richtung.

c) Farbdefinition für Füllungen: **SetOPen**
Oben habe ich im Zusammenhang mit dem Flood-Befehl von dem Modus gesprochen, bei dem solange gefüllt wird, bis auf eine bestimmte Farbe getroffen wird, die nicht mit der Zeichenfarbe identisch sein muß. Man kann dafür ebenfalls ein Makro aufstellen, das die Farbregisternummer in die Rastport-Struktur einträgt.

```
SetOPen        macro
               move.w d0,$1B(a1)
               move.w $20(a1),d0
               or.w #8,d0
               move.w d0,$20(a1)
               endm
```

Neben der Registernummer ist auch der Eintrag eines Flags erforderlich. Hier der Aufruf dieser Funktion:

```
move.l RP,a1                 ;Zeiger auf Rastport
move.l farbe,d0              ;Farbregisternummer
SetOPen                      ;Funktion aufrufen
```

4.5 Die Benutzung von Disketten- und ROM-Zeichensätzen

Neben den mächtigen Grafikfunktionen stellt uns der Amiga auch eine enorme Schriftenvielfalt zur Verfügung. Wenn Sie einmal auf Ihrer Workbench-Diskette das Unterdirectory »Fonts« ansehen, bekommen Sie einen ersten Eindruck. Leider wird die Möglichkeit der Zeichensatzmodifikation aber viel zu selten genutzt. Ein Grund dafür mag sein, daß der Zugriff auf die Disketten- und ROM-Zeichensätze (davon gibt es zwei) relativ kompliziert ist, besonders dann, wenn die Namen und Anzahl der Zeichensätze nicht bekannt ist. Prinzipiell ist folgendes Vorgehen erforderlich:

a) Verfügbare Zeichensätze auf Disk und im ROM ermitteln.
b) Mit OpenFont (ROM) bzw. OpenDiskFont (Disk) müssen sie bereitgestellt werden.
c) Mit AddFont werden sie für das System verfügbar gemacht.
d) Mit SetFont wird ein Zeichensatz aktiviert.
e) Mit AskSoftStyle und SetSoftStyle kann seine Darstellungsweise verändert werden (Unterstreichen, Kursivschrift, etc.).
f) Mit Text können nun Zeichen ausgegeben werden.
g) Mit RemFont können die Zeichensätze dem System entzogen werden.
h) Mit CloseFont können die Zeichensätze aus dem Speicher entfernt werden.

Sie sehen, liebe Leser, daß eine Menge an Arbeit vor uns liegt, bevor wir unsere Textausgaben bewundern können. Die Bilder 4.1. und 4.2, die unserem nächsten Beispielprogramm entnommen sind, zeigen aber, was für schöne Schriften man auf den Bildschirm zaubern kann.

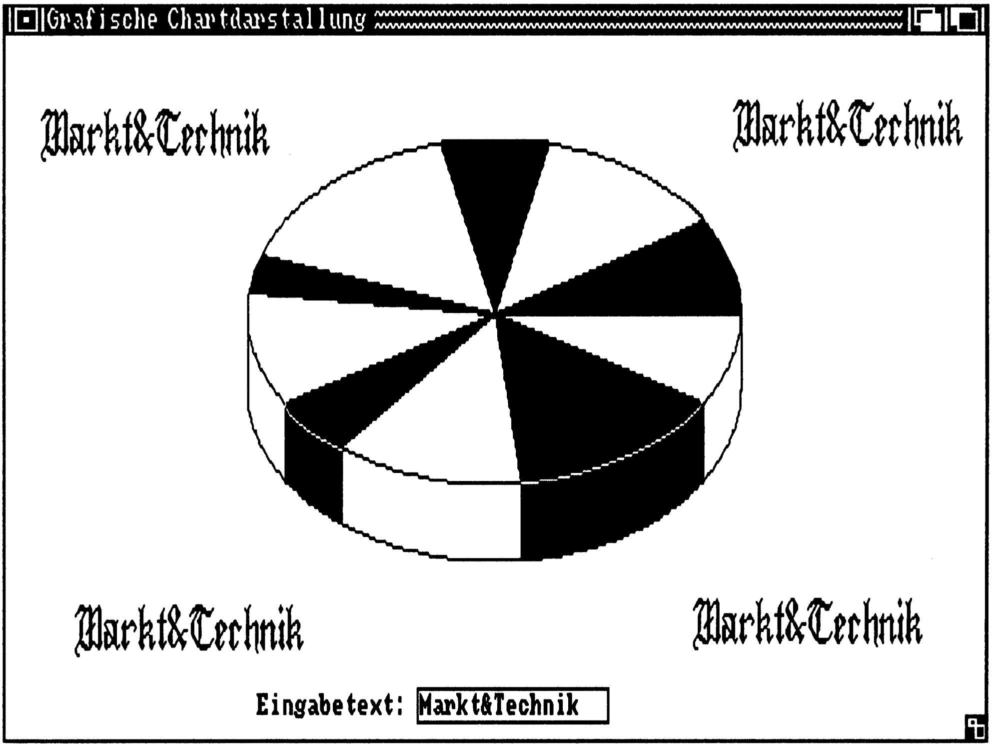


Bild 4.1: Tortengrafik beschriftet mit SAPHIRE 19

4.5.1 Zeichensätze laden und aktivieren

a) Zeichensätze ermitteln: **AvailFont (Diskfont-Library)**
Falls die Anzahl der zur Verfügung stehenden Zeichensätze sowie deren Namen nicht bekannt sind, ist der Einsatz dieser Funktion angezeigt. Sie ermittelt selbständig sämtliche Disk- und ROM-Zeichensätze. Dazu muß man einen Puffer reservieren, in den als erstes Wort die Gesamtanzahl der gefundenen Zeichensätze eingetragen werden. Daran schließt sich für jeden Zeichensatz eine Struktur an, die zunächst in einem weiteren Wort angibt, ob es sich um ein ROM(1)- oder um einen Disk(2)-Font handelt. Danach kommt die sogenannte TextAttr-Struktur, die einen Zeiger auf den Zeichensatznamen, die Größe, den Style und ein Flag enthält. Diese Struktur wird für das weitere Vorgehen noch benötigt. Wenn die Zeichensätze aber bekannt sind (wie in

unserem zweiten Beispielprogramm), kann man die TextAttr-Struktur auch per Hand anlegen. Darauf kommen wir noch zurück. Zunächst möchte ich Ihnen jedoch an Hand eines Beispiels ein mögliches Ergebnis der Avail-Font-Funktion aufzeigen:

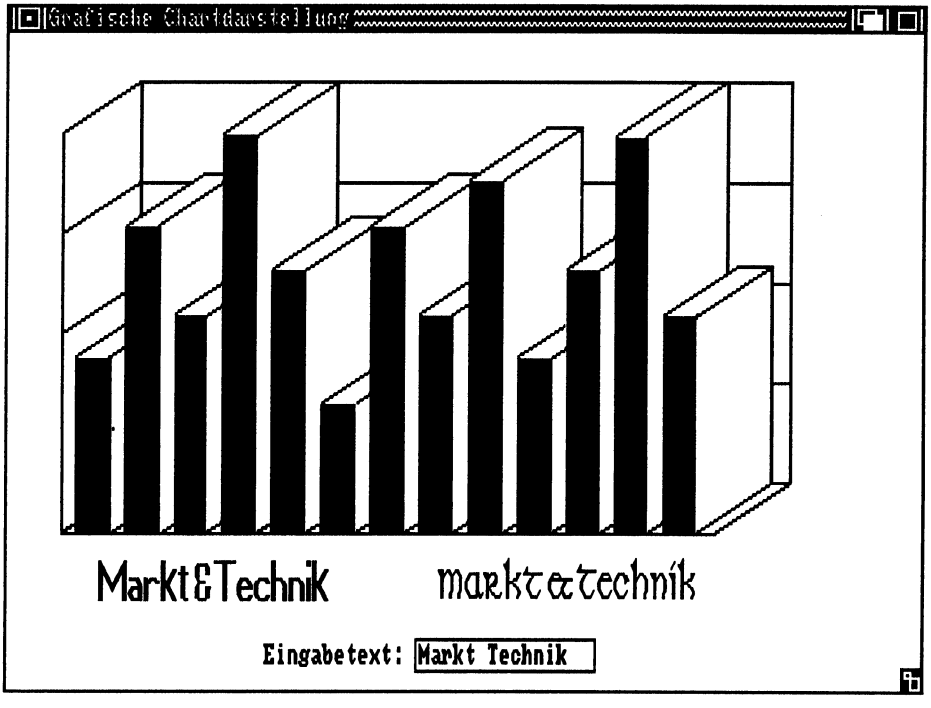


Bild 4.2: Säulengrafik beschriftet mit DIAMOND 20 und ESMERALD 20

Das Problem bei unbekannten Zeichensätzen besteht darin, daß man sie nicht im Menü anbieten kann, es sei denn, man generiert innerhalb seines Programms eine Menüstruktur abhängig von den gefundenen Zeichensätzen. Dies ist theoretisch möglich, erfordert aber einen riesigen Programmieraufwand. Es ist deshalb günstiger, alle benötigten Zeichensätze auf die Programmdisk zu kopieren und die TextAttr-Strukturen fest anzulegen. Damit sind dann auch die Anzahl und Inhalte etwaiger Menüpunkte zur Zeichensatzanwahl bekannt.

Adresse	Inhalt	
puffer	dc.w 4	;2 Zeichensätze gefunden
Es folgen die Einträge für den ersten Zeichensatz:		
puffer+2	dc.w 1	;Zeichensatz im ROM
Hier beginnt die TextAttr-Struktur des ersten Zeichensatzes:		
puffer+4	dc.l z1	;Zeiger auf Zeichensatzname
puffer+8	dc.w 8	;Größe: 8 Pixel
puffer+10	dc.b 0	;Normaldarstellung
puffer+11	dc.b $49	;Flags
Es folgen die Einträge für den zweiten Zeichensatz:		
puffer+12	dc.w 2	;Zeichensatz auf Disk
Hier beginnt die TextAttr-Struktur des zweiten Zeichensatzes:		
puffer+14	dc.l z2	;Zeiger auf Zeichensatzname
puffer+18	dc.w 12	;Größe: 8 Pixel
puffer+20	dc.b 0	;Normaldarstellung
puffer+21	dc.b $62	;Flags
puffer+22	z1	dc.b "topaz.font",0
puffer+32	z2	dc.b "ruby.font",0

Wenn man später die Zeichensätze laden möchte, ist der jeweilige Zeiger auf die TextAttr-Struktur erforderlich. Diesen kann man nach der Formel

Textattr = puffer + 4 + (Zeichensatznummer-1)*10

berechnen, z.B. ergibt sich für den fünften gefundenen Zeichensatz die Adresse puffer + 44.

Beim Aufruf der AvailFont-Funktion muß neben der Startadresse des Puffers und dessen Größe auch ein Flag übergeben werden, indem man festlegen kann, ob die Zeichensätze nur im ROM(1), nur auf der Disk(2) oder auf beiden Elementen (3) gesucht werden sollen.

```
move.l #puffer,a0               ;Startadresse Puffer
move.l #größe,d0                ;Größe Puffer in Bytes
move.l #flag,d1                 ;ROM oder Disk (oder beides)
CALLDISKFONT AvailFonts         ;Zeichensätze suchen
```

Etwas problematisch ist die Festlegung der Größe des Puffers, da die benötigte Größe natürlich von der Zahl der gefundenen Zeichensätze abhängt. Weil man diese aber vorher noch nicht kennt, kann man die benötigte Größe nicht vorhersagen. Pro vermutetem Zeichensatz muß man mindestens 20 Byte reservieren (von der Länge des Namens abhängig). Wenn mehr Zeichensätze als vermutet gefunden werden, können diese leider nicht mehr berücksichtigt werden. Am einfachsten ist es aber, von Anfang an z.B. 1 Kilobyte für den Puffer zu reservieren.

b) Disk-Zeichensätze einlesen: **OpenDiskFont (Diskfont-Library)**
Nachdem man alle TextAttr-Strukturen vorliegen hat, kann man beginnen, die gewünschten Zeichensätze in den Speicher zu holen. Fatal wäre es allerdings, den Versuch zu starten, einen ROM-Zeichensatz einzuladen. Bevor man daher diese Funktion einsetzt, muß man zunächst prüfen, ob es sich wirklich um einen Disk-Font handelt. Übergeben wird der Zeiger auf die TextAttr-Struktur, als Ergebnis bekommt man einen weiteren Zeiger auf die sogenannte TextFont-Struktur. Diese wird benötigt, wenn man den Zeichensatz für das System verfügbar machen möchte, durch die OpenDiskFont-Funktion wird er nur geladen.

```
move.l #textattr,a0                 ;Zeiger auf TextAttr-Struktur
CALLDISKFONT OpenDiskFont           ;Zeichensatz laden
move.l d0,textfont                  ;Pointer retten
textfont   dc.l 0
```

c) ROM-Zeichensatz einlesen: **OpenFont**
Etwas merkwürdig erscheint, daß auch ein Zeichensatz »geladen« werden muß, der sowieso schon im Speicher steht. Einsichtig ist aber zumindest die Übergabe des Zeigers auf die TextFont-Struktur.

```
move.l #textattr,a0                 ;Zeiger auf TextAttr-Struktur
CALLGRAF OpenFont                   ;Zeichensatz laden
move.l d0,textfont                  ;Pointer retten
textfont   dc.l 0
```

d) Aktuellen Zeichensatz retten: **AskFont**
Wenn man mit neuen Zeichensätzen arbeitet, muß man sich darüber im klaren sein, daß die Daten des geänderten Zeichensatzes verloren sind, so daß man ihn am Ende des Programms nicht wieder mit SetFont restaurieren kann. Ein gutes Programm wird deshalb zunächst, bevor die erste Änderung vollzogen wird, die Daten des aktiven Zeichensatzes retten. Dies geschieht mit der AskFont-Funktion, die eine TextAttr-Struktur mit dessen Daten anlegt. Mit deren Hilfe kann man dann durch die OpenFont-Funktion eine TextFont-Struktur generieren.

```
move.l RP, a1                       ;Zeiger auf Rastport
move.l #platz,a0                    ;Speicher für TextAttr-Struktur
#CALLGRAF AskFont                    ;Struktur generieren
platz   dc.b 8                      ;8 Byte reservieren
```

4.5.2 Zeichensätze aktivieren

Von nun an wird kein Unterschied mehr zwischen ROM- und Disk-Zeichensätzen gemacht, da sich alle im Speicher befinden und nur noch durch die TextFont-Struktur gekennzeichnet sind.

a) Zeichensatz verfügbar machen : **AddFont**
Durch diese Funktion wird der entsprechende Zeichensatz an die schon bestehende Zeichensatzliste im System angehängt. Normalerweise steht dort nur ein Eintrag, nämlich der Standardzeichensatz. Sie können ihn nun über den Zeiger auf die Textfont-Struktur z.B. als Grundlage für Screens oder Windows heranziehen, indem Sie den Zeiger in die NewWindow bzw. NewScreen-Struktur eintragen.

```
move.l textfont,a1              ;Zeiger auf Textfont-Struktur
CALLGRAF AddFont
```

b) Zeichensatz setzen : **SetFont**
Hiermit wird einem bestimmten Rastport der durch den Zeiger auf die Textfont-Struktur gekennzeichnete Zeichensatz zugeordnet, d.h., alle weiteren Textausgaben in diesen Rastport erfolgen mit dem neuen Zeichensatz.

```
move.l RP,a1                    ;Zeiger auf Rastport
move.l textfont,a0              ;Zeiger auf TextFont-Struktur
CALLGRAF SetFont
```

4.5.3 Modifikation des Styles

Wer nun glaubt, mit der Aktivierung eines neuen Zeichensatzes seien schon alle Möglichkeiten erschöpft, täuscht sich gewaltig. Man kann nämlich das Aussehen der einzelnen Zeichen auch innerhalb eines Zeichensatzes ändern. Dafür existiert ein Byte, dessen unterste 4 Bit jeweils einen Style repräsentieren:

Bit	Wert	Style
0	1	Unterstrichen
1	2	Fettdruck
2	4	Schrägschrift
3	8	Breitschrift

Im Normalfall ist keines der Bits gesetzt. Man muß allerdings sagen, daß nicht alle Styles bei allen Zeichensätzen Wirkung zeigen. Am besten, Sie probieren es einfach mal mit dem Beispielprogramm »Chart-Grafik« aus, zu dem wir nach diesem Abschnitt kommen werden.

a) Ermittlung der momentanen Zeichensatzeinstellung : **AskSoftStyle**
Normalerweise betrachte ich Funktionen zur Ermittlung irgendeiner Einstellung nicht als so wichtig, in diesem Fall wird das Ergebnis aber für eine Neueinstellung benötigt. Hierbei handelt es sich um ein Byte, in dem die aktivierten Styles durch ein gesetztes Bit dargestellt werden, z.B. würde eine Fünf bedeuten, daß unterstrichen und schräg geschrieben wird.

```
move.l RP,a1                    ;Rastport holen
CALLGRAF AskSoftStyle           ;Maske holen
move.l d0,maske                 ;und sichern
```

b) Neusetzen des Darstellungsmodus : SetSoftStyle
Durch die Übergabe der neuen Maske wird der Style des Zeichensatzes verändert. Dazu muß aber auch der alte Wert, den man von der Funktion AskSoftStyle erhalten hat, übergeben werden, da aus diesen beiden Werten die neue Maske berechnet wird.

```
move.l RP,a1                    ;Rastport holen
move.l maske,d1                 ;alte Maske holen
move.l #neumask,d0              ;neue Maske holen
CALLGRAF SetSoftStyle           ;neue Styles setzen

neumask  dc.b %00001100         (z.B. Schräg- und Breitschrift ein)
```

4.5.4 Zeichensätze löschen

a) Zeichensatz aus Liste löschen : RemFont
Der Zeichensatz steht nach dieser Funktion dem System nicht mehr zur Verfügung, wobei er natürlich gleichzeitig deaktiviert wird. Um ihn wieder benutzen zu können, müssen die Funktionen AddFont und SetFont aufgerufen werden. Um einen Zeichensatz zu deaktivieren, ohne ihn aus der Liste zu streichen, genügt es, mit der SetFont-Funktion den neuen Zeichensatz einzuschalten.

```
move.l textfont,a1              ;Zeiger auf Textfont-Struktur
CALLGRAF RemFont
```

b) Zeichensatz aus Speicher entfernen: CloseFont
Mit dieser Funktion wird der Zeichensatz nicht nur gelöscht, sondern ganz aus dem Speicher eleminiert. Erforderlich ist diese Prozedur immer am Programmende oder wenn Speicherprobleme auftreten. Ein auf diese Weise gelöschter Zeichensatz muß zur erneuten Benutzung erst mit OpenFont bzw. OpenDiskFont geladen werden.

```
move.l textfont,a1              ;Zeiger auf TextFont-Struktur
CALLGRAF CloseFont              ;Zeichensatz löschen
```

Damit sind wir am Ende der Grafik- und Zeichensatzfunktionen angekommen. Man sieht, daß es gar nicht so schwer ist, seinen Programmen Elemente mitzugeben, die ihnen einen professionellen Touch verleihen. In dem nun folgenden Beispielprogramm »Chart-Grafik« werden Sie alle wesentlichen Funktionen, die wir eben besprochen haben, wiederfinden.

4.6 Programmierung einer Chart-Grafik

Unser zweites Beispielprogramm behandelt ein in Geschäftskreisen sehr beliebtes Thema, die sogenannten Chart-Grafiken. Hierbei handelt es sich um Diagramme, mit denen man Umsätze, Verkaufszahlen, etc. grafisch veranschaulichen kann. Ein weiteres Einsatzgebiet, das Sie alle kennen, ist die Darstellung von Wahlergebnissen in Form von Prozenten und Sitzverteilungen. Insgesamt kann man zwischen drei Gruppen der Grafiken unterscheiden:

- Kuchen- und Tortendiagramme (zwei- oder dreidimensional)
- Balken- und Säulendiagramme (zwei- oder dreidimensional)
- Kurvendiagramme (zweidimensional)

Wenn Sie ein Umsteiger vom C64 sind, liebe Leser, werden Sie vielleicht das Programm »Grafik-Calc 64« kennen, das das leistungsfähigste seiner Art ist und sieben verschiedene Grafiktypen erzeugen kann. Sie werden einsehen, daß der Programmieraufwand für ein solches Programm die Möglichkeiten dieses Buches übersteigt. Wir werden uns daher auf eine dreidimensionale Tortengrafik und eine Säulengrafik beschränken.

Ein Chart-Programm benötigt neben den Grafikroutinen noch einen Eingabemodus, in dem die darzustellenden Werte eingegeben werden und sinnvollerweise auch Routinen zum Abspeichern und Einladen der Daten. Letzteres kommt eigentlich zu früh, da wir die DOS-Funktionen erst im nächsten Kapitel behandeln werden, aber ohne sie wäre das Programm nur die Hälfte wert. Wenn man die Grafiken erstellt hat, möchte man sie meistens auch noch beschriften, um den einzelnen Tortenstücken oder Säulen auch optisch den entsprechenden Datensatz zuordnen zu können. Programme wie »Grafik-Calc 64« erledigen dies selbstverständlich automatisch, wir wollen uns aber damit begnügen, per Mausdruck einen vorher festgelegten Text in die Grafik schreiben zu können. Das Aussehen des Textes wollen wir zudem möglichst variabel gestalten, indem wir zwei Menüs für eine Wahl des Zeichensatzes und der Schriftart bereithalten.

Wenn Sie das Programm »Chart-Grafik.obj« von der Workbench oder dem CLI gestartet haben, dauert es erst einmal eine Weile, bis alle Zeichensätze aus dem ROM oder von der Diskette geladen wurden. Wenn das Startbild erscheint, wählen Sie bitte zunächst im Daten-Menü den Punkt »Eingeben« an. Sie können dann in dem Eingabewindow 20 Datensätze eingeben, was wiederum durch String-Gadgets realisiert wurde. Sie können entweder die Eingabe beenden oder weitere Datensätze eingeben, worauf weitere Eingabefelder bereitgestellt werden. Insgesamt können Sie maximal 100 Datensätze eingeben.

Falls Sie die Daten nochmals benötigen, sollten Sie sie in dem Menüpunkt »Speichern« auf der Diskette ablegen. Dazu tragen Sie einfach den Filenamen in das Eingabefeld ein und klicken die Funktion »Ausführen« an. Später können Sie die Daten dann im Menüpunkt »Laden« auf die gleiche Art und Weise wieder einladen. Falls beim Diskettenzugriff etwas schiefgeht, erscheint ein Requester, wonach Sie wahlweise einen erneuten Versuch starten können oder den Abbruch der Funktion erzwingen können.

Als nächstes müssen Sie festlegen, welche der eingegebenen oder geladenen Datensätze Sie grafisch darstellen möchten. Dazu geben Sie den Start- und Enddatensatz in die

vorbereiteten Gadgets ein. Danach kann es losgehen: Durch Anklicken entweder der Torten- oder Säulengrafik wird ein Screen im HiRes-Modus geöffnet und die Grafik aufgebaut. Dabei fällt auf, wie extrem schnell die Säulen aufgebaut und die Tortenstücke ausgefüllt werden, da hierfür die Area-Befehle benutzt werden. Der Aufbau der Ellipsen für die Torte ist dagegen sehr langsam, was im Fehlen einer vernünftigen Ellipsen-Routine begründet ist.

Am unteren Rand sehen Sie ein weiteres Stringgadget, in das Sie den Text eintragen müssen, mit dem Sie die Grafik beschriften möchten. Nach der Eingabe fahren Sie einfach mit der Maus an die gewünschte Position und drücken die linke Maustaste, worauf der Text ausgegeben wird. Falls Ihnen der normale Zeichensatz nicht ausreicht, können Sie mit Hilfe der Menüs »zeichensatz« und »Style« das Aussehen der Zeichen modifizieren. Die Menüpunkte des Menüs »Style« werden dabei wechselweise an- und abgewählt, was durch das Abhaken verdeutlicht wird. Wenn ein neuer Zeichensatz angewählt wurde, werden automatisch alle Styles zurückgesetzt, da nicht angenommen wird, daß Sie diejenigen des vorher aktiven Zeichensatzes übernehmen möchten. Verlassen wird der Grafikmodus durch Anklicken des Close-Gadgets. Das Hauptprogramm kann mit der Option »Ende« beendet werden.

Dieses Programm bietet, ähnlich wie die Wertetabelle, für ein Beispielprogramm innerhalb eines Buches wieder sehr viel, vor allen Dingen werden erneut die Dinge in der Praxis demonstriert, deren Funktion Sie bislang nur theoretisch kannten. Der Quelltext ist daher auch recht umfangreich, ich habe ihn aber wieder in kleine Stücke zerlegt, die einzeln kommentiert werden. Überflüssige Funktionen der Include-Files wurden eleminiert, um Platz zu sparen.

Damit das Programm richtig funktionieren kann, ist es erforderlich, die Zeichensätze der Workbench-Diskette auf Ihre DEVPAC-Arbeitsdiskette zu kopieren, wie dies in Kapitel 1 nachzulesen ist.

4.6.1 Include-Files und Makrodefinitionen

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
            opt l-,d+,s-,n+
            plen 44
            llen 80

* INCLUDE-Files laden

        include df0:include/exec/exec_lib.i
        include df0:include/math/mathffp_lib.i
        include df0:include/intuition/intuition_lib.i
        include df0:include/math/mathtrans_lib.i
        include df0:include/graphics/graphics_lib.i
```

```
  include df0:include/libraries/dos_lib.i
  include df0:include/libraries/diskfont_lib.i

* MACRO-Definitionen

* Intuition-Text ausgeben

PRINTOUT        macro
                lea \1,a1
                move.l #\2,d0
                move.l #\3,d1
                move.l windowpointer,a0
                move.l 50(a0),a0
                CALLINT PrintIText
                endm

*Struktur für Ausgabe-Text

TEXTOUT         macro
                dc.b 3,0          ;Farben
                dc.b  0           ;Farbmodus
                even
                dc.w  0           ;x-Koordinate
                dc.w  0           ;y-Koordinate
                dc.l  0           ;Zeichensatz
                dc.l  \1          ;Textpointer
                dc.l  0           ;weiterer Text
                endm

TEXTOUT1        macro
                dc.b 0,0          ;Farben
                dc.b  0           ;Farbmodus
                even
                dc.w \1           ;x-Koordinate
                dc.w \2           ;y-Koordinate
                dc.l  0           ;Zeichensatz
                dc.l \3           ;Textpointer
                dc.l  0           ;weiterer Text
                endm

*Struktur für Menü

MENUE           macro
                dc.l \1          ;nächstes Menü
                dc.w \2,\3       ;x,y des Menütitels
                dc.w 70,10       ;Breite, Höhe des Menütitels
                dc.w 1           ;Menü wählbar
```

```
                dc.l \4         ;Menütitel
                dc.l \5         ;Menüeinträge
                dc.w 0,0,0,0
                endm

*Struktur für Menüpunkte

MENUPUNKT       macro
                dc.l  \1        ;nächster Menüpunkt
                dc.w  \2,\3     ;x,y-Position
                dc.w  \4        ;Breite
                dc.w  10        ;Höhe
                dc.w  \5        ;Flag
                dc.l 0          ;kein Ausschluß
                dc.l  \6        ;Zeiger auf Textstruktur
                dc.l  0         ;keine Zeichnung
                dc.b  \7        ;Amiga-Taste
                even
                dc.l  \8        ;Zeiger auf Untermenü
                dc.l  0
                endm

* Struktur für Windows

WINDOW       macro
             dc.w       \1      ;linke Ecke
             dc.w       \2      ;obere Ecke
             dc.w       \3      ;Breite
             dc.w       \4      ;Höhe
             dc.b       0,1     ;Farben
             dc.l       \5      ;Flags für Events
             dc.l       \9      ;Window Flags
             dc.l       \6      ;Zeiger auf erstes Gadget
             dc.l       0       ;Check Mark
             dc.l       \7      ;Windowname
\8           dc.l       0       ;Screenpointer
             dc.l       0       ;Bit Map
             dc.w      \3       ;Mindest-Breite
             dc.w      \4       ;Mindest-Höhe
             dc.w      \3       ;Maximale Breite
             dc.w      \4       ;Maximale Höhe
             dc.w      15       ;Window-Typ
             endm
```

```
* Struktur für Screens

SCREEN          macro
           dc.w       0          ;linke Ecke
           dc.w       0          ;obere Ecke
           dc.w       \1         ;Breite
           dc.w       \2         ;Höhe
           dc.w       2          ;Tiefe (Bitplanes)
           dc.b       2,3        ;Farben
           dc.w       \3         ;Modus
           dc.w      15          ;Screentyp
           dc.l       0          ;Zeichensatz
           dc.l       \4         ;Screentitel
           dc.l       0          ;Gadgets
           dc.l       0          ;Bitmap
                endm

* Struktur für Gadgets

GADGET     macro
           dc.l       \1         ;nächstes Gadget
           dc.w       \2         ;obere Ecke x
           dc.w       \3         ;obere Ecke y
           dc.w      122         ;Breite
           dc.w       10         ;Höhe
           dc.w        0         ;Flags
           dc.w       \4         ;Activation-Flags
           dc.w       \5         ;Gadget-Typ
           dc.l       \6         ;Gadget-Border
           dc.l        0         ;kein Select-Gadget
           dc.l       \7         ;Gadget-Text
           dc.l        0         ;kein Exclude
           dc.l       \8         ;Special-Info
           dc.w       \9         ;Gadget-ID
           dc.l        0         ;User-Data
           endm

* Struktur für Eingaben bei Stringgadgets

STRINF     macro
           dc.l  \1              ;Textpuffer
           dc.l  undo            ;Undo-Puffer
           dc.w  0               ;Cursor-Position
           dc.w  15              ;Anzahl Zeichen (max)
           dc.w  0               ;Ausgabeposition Cursor
           dc.w  0               ;Position Undo-Puffer
           dc.w  0               ;Anzahl Zeichen im Textpuffer
```

```
            dc.w  0               ;Anzahl sichtbare Zeichen
            dc.w  0               ;horizontaler Offset
            dc.w  0               ;vertikaler Offset
            dc.l  0               ;Zeiger auf Rastport
            dc.l  0               ;Integer-Wert
            dc.l  0               ;Tastatur-Belegung
            endm

* Textstruktur für Gadgets

STRING      macro
            dc.b  1,0             ;Farben
            dc.b  1               ;Farbmodus
         even
            dc.w  \1,\2           ;Positionen
            dc.l  0               ;Zeichensatz
            dc.l  \3              ;Textpointer
            dc.l  \4              ;nächster Text
            endm

* Puffer für Stringgadets

ZAHLEN      macro
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            dc.b "0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            endm

* Zeichensatz

ZEICHEN     macro
            dc.l \1                ;Pointer auf Name
            dc.w \2                ;Größe in Pixeln
            dc.b \3                ;Style
            dc.b \4                ;Flag
            endm


0000006E undo         dc.l  0,0,0,0          ;Undo-Puffer für Stringgadgets
```

Diesen Programmabschnitt kennen Sie größtenteils schon von der Wertetabelle. Neu hinzugekommen ist lediglich das Makro »ZEICHEN«, welches für die TextAttr-Struktur verwendet wird. Da dieses Programm die Zeichensätze nicht mit AvailFonts ermittelt, sondern auf bestimmte als vorhanden angenommene Zeichensätze zugreift, müßten die TextAttr-Strukturen vom Programmierer ausgefüllt werden, was bei 15 Zeichensätzen ein relativ großer Aufwand wäre. Obwohl das Makro relativ kurz ist, bedeutet es eine deutliche Verkürzung des Quelltextes und der Programmierzeit.

4.6.2 Öffnen der Librarys, Zeichensätze und Grafik, Menüauswertung

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
0000007E begin    lea ffpname,a1              ;Math-Library öffnen
00000084          clr.l d0
00000086          CALLEXEC OpenLibrary
0000008E          tst.l d0
00000090          beq fini1
00000094          move.l d0,_MathBase

0000009A          lea intname,a1              ;Intuition-Library öffnen
000000A0          clr.l d0
000000A2          CALLEXEC OpenLibrary
000000AA          tst.l d0
000000AC          beq fini2
000000B0          move.l d0,_IntuitionBase

000000B6          lea mathtransname,a1        ;Mathtrans-Library öffnen
000000BC          clr.l d0
000000BE          CALLEXEC OpenLibrary
000000C6          tst.l d0
000000C8          beq fini3
000000CC          move.l d0,_MathTransBase

000000D2          lea grafname,a1             ;Grafik-Library öffnen
000000D8          clr.l d0
000000DA          CALLEXEC OpenLibrary
000000E2          tst.l d0
000000E4          beq fini4
000000E8          move.l d0,_GfxBase

000000EE          lea dosname,a1              ;DOS-Library öffnen
000000F4          clr.l d0
000000F6          CALLEXEC OpenLibrary
000000FE          tst.l d0
00000100          beq fini5
```

```
00000104        move.l d0,_DOSBase
0000010A        lea diskfontname,a1            ;Diskfont-Library öffnen
00000110        clr.l d0
00000112        CALLEXEC OpenLibrary
0000011A        tst.l d0
0000011C        beq fini6
00000120        move.l d0,_DiskfontBase

00000126        lea screen_defs,a0             ;Screen öffnen
0000012C        CALLINT OpenScreen
00000136        move.l d0,screenpointer        ;Pointer für
0000013C        move.l d0,point1
00000142        move.l d0,point2               ;windows
00000148        move.l d0,point3

        * Zeichensätze holen

0000014E        bsr opdifo                     ;von Diskette
00000152        bsr opfo                       ;aus dem ROM
00000156        bsr adfo                       ;Zeichensätze holen

0000015A        move.l #640,d0                 ;Speicherplatz für
00000160        move.l #256,d1                 ;FLOOD bereitstellen
00000166        CALLGRAF AllocRaster           ;(Breite,Höhe Bildschirm)
00000170        move.l d0,mempoint             ;Pointer sichern

00000176        lea areainf,a0                 ;Platz für Area-Infos
0000017C        lea platz,a1                   ;Platz für Koordinaten
00000182        move.l #10,d0                  ;max. 10 Einträge
00000188        CALLGRAF InitArea              ;Area initialisieren

00000192        move.l mempoint,a1             ;Pointer auf Puffer
00000198        move.l #40960,d0               ;Größe in Bytes
0000019E        lea tmpras,a0                  ;Start Puffer
000001A4        CALLGRAF InitTmpRas            ;Puffer initialisieren

        * Menüaufbau, -abfrage

000001AE        lea startwindow,a0             ;Startwindow öffnen
000001B4        CALLINT OpenWindow
000001BE        move.l d0,windowpointer

000001C4        bsr print

000001C8        move.l windowpointer,a0
000001CE        lea menu,a1
000001D4        CALLINT SetMenuStrip           ;Menü setzen
```

```
000001DE          move.l windowpointer,a0       ;aktuellen Zeichensatz
000001E4          move.l 50(a0),a1
000001E8          lea zeichenleer,a0            ;sichern
000001EE          CALLGRAF AskFont
000001F8          lea zeichenleer,a0            ;in leere Struktur
000001FE          CALLGRAF OpenFont
00000208          lea leer,a0
0000020E          move.l d0,(a0)
00000210          bra wait11

00000214 wait     move.l windowpointer,a0       ;Startzeichensatz
0000021A          move.l 50(a0),a1
0000021E          lea leer,a0                   ;reaktivieren
00000224          move.l (a0),a0
00000226          CALLGRAF SetFont


00000230 wait11   move.l windowpointer,a0
00000236          move.l 86(a0),a0              ;User-Port holen
0000023A          move.l a0,a5                  ;Adresse retten
0000023C          move.b 15(a0),d1              ;Signal-Bit holen
00000240          moveq #0,d0                   ;Nummer in Maske
00000242          bset d1,d0                    ;wandeln
00000244          CALLEXEC Wait                 ;Auf Wiedersehen!
0000024C          move.l a5,a0
0000024E          CALLEXEC GetMsg               ;Message holen
00000256          move.l d0,a1
00000258          clr.l d7
0000025A          move.w $18(a1),d7             ;Menüpunkt holen
0000025E          move.l $1C(a1),d6             ;Adresse Gadget holen
00000262          move.l $14(a1),d5             ;IDCMP-Flag holen
00000266          CALLEXEC ReplyMsg             ;Message quittieren
0000026E          cmpi.l #$100,d5               ;Menüpunkt?
00000274          beq menü                      ;Ja!

         * Kein Menüpunkt, also Gadget angewählt

00000278          move.l d6,a2
0000027A          move.w $26(a2),d2             ;ID aus Gadget-Struktur
0000027E          cmpi.w #111,d2
00000282          beq torte                     ;Tortengrafik
00000286          cmpi.w #112,d2
0000028A          beq saeule                    ;Säulengrafik
0000028E          bne wait11                    ;Datensatz-Gadget
```

```
         * Behandlung Menüpunkte

00000292 menü     cmpi.w #$ffff,d7                ;Keiner
00000296          beq waitl1
0000029A          bsr wert                        ;Menüpunkt auswerten
0000029E          lea tab,a4
000002A4 spring   adda.l d5,a4
000002A6          move.l (a4),a4                  ;Menüpunkt feststellen
000002A8          adda.l d6,a4
000002AA          move.l (a4),a4
000002AC          jmp (a4)                        ;und anspringen

000002AE tabmenul          dc.l endof                        ;Menüpunkt Menül
000002B2 tabmenu2          dc.l eingeben,laden,speichern     ;Menüpunkte Menü2

000002BE tab               dc.l tabmenul,tabmenu2            ;Menüs 1 bis 2

         * Menüpunkt auswerten

000002C6 wert     move.w d7,d6
000002C8          lsr #7,d7
000002CA          lsr #4,d7
000002CC          lsl #2,d7                       ;Untermenü*4 in D7
000002CE          move.w d6,d5
000002D0          andi.l #%00000000000000000000011111100000,d6
000002D6          lsr #3,d6                       ;Menüpunkt*4 in D6
000002D8          andi.l #%00000000000000000000000000011111,d5
000002DE          lsl #2,d5                       ;Menü*4 in D5
000002E0          rts
```

Zunächst werden alle Ihnen schon von der Wertetabelle bekannten Libraries geöffnet. Neu dazugekommen ist die Diskfont-Library, die für die OpenDiskFont-Funktion benötigt wird. Anschließend wird der Textscreen in der Auflösung von 320 * 256 Punkten geöffnet und der Pointer auf die Screen-Struktur in die Makros der Windows eingetragen. Anschließend werden in drei Unterprogrammen die Zeichensätze von der Diskette und aus dem ROM geholt sowie dem System verfügbar gemacht. Nun folgt der Teil, in dem Speicherplatz reserviert und für die Area-Befehle initialisiert wird. Da unser Grafikschirm eine Auflösung von 640 * 256 Punkten bei der Verwendung von zwei Bitplanes bietet, müssen wir (640*256*2)/8 = 40 960 Byte reservieren. Jetzt kann das Startwindow geöffnet und das Menü installiert werden.

Interessant ist der nun folgende Teil: Da wir mit mehreren Zeichensätzen operieren wollen, müssen wir die Daten des aktuellen Zeichensatzes retten, wenn wir unser Programm mit dem gleichen Zeichensatz verlassen wollen, mit dem wir es gestartet haben. Dazu müssen eine TextAttr- und eine TextFont-Struktur eingerichtet werden. Die TextAttr-Struktur kann

man durch die AskFont-Funktion erhalten, wie wir es oben besprochen haben. Dazu ist der Zeichensatz-Pointer aus dem Rastport erforderlich, der ab der Adresse dez. 50 in der Rastport-Struktur zu finden ist. Die Daten sollen ab dem Label »zeichenleer« abgelegt werden. Wenn diese Struktur angelegt wurde, können wir sie gleich dazu benutzen, auch die TextFont-Struktur mit Hilfe der OpenFont-Funktion einzurichten. Der Zeiger wird ab dem Label »leer« gespeichert. Die Frage ist, warum wir diese Strukturen überhaupt benötigen. Nun, um den Zeichensatz, der am Anfang unseres Programms aktiv war, auch am Ende wieder aktivieren zu können, müssen wir die Funktion SetFont benutzen. Wie wir wissen, benötigt sie die TextFont-Struktur. Diese erhalten wir aber nur aus der OpenFont-Funktion, die ihrerseits die TextAttr-Struktur benötigt. An sich würden wir diese Struktur nicht brauchen, wir erzeugen sie nur, um die OpenFont Funktion überhaupt benutzen zu können.

Als nächstes haben wir dann die Auswertung des Message-Ports. Im Unterschied zu der Wertetabelle muß hier nicht nur darauf geprüft werden, welcher Menüpunkt gewählt wurde, sondern auch, ob überhaupt ein solcher angewählt wurde, da ja noch vier Gadgets zu bedienen sind. Wenn das IDCMP-Flag kein Menü anzeigt, wird als nächstes untersucht, ob eines der beiden Hauptmenü-Gadgets angeklickt wurde und entsprechend zu den Routinen für die Torten- und Säulengrafik verzweigt. Falls keines dieser beiden Gadgets gemeint war, was daran zu erkennen ist, daß die ID-Nummern nicht übereinstimmen, handelte es sich um das Start- oder Endwert-Gadget, welches für uns (noch) keine Rolle spielt. Die Auswertung des Menüpunktes erfolgt analog zur Wertetabelle. Weil wir im Grafikmodus aber noch ein zweites Menü bekommen, wurde die eigentliche Auswertung des Codewortes in ein Unterprogramm verlagert, das dann auch von diesem Menü aus genutzt werden kann. Damit ist das eigentliche Hauptprogramnm schon wieder beendet, es folgen die Routinen der einzelnen Menüpunkte.

4.6.3 Daten eingeben

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * ERSTES MENÜ (DATEN)

         * Menüpunkt Daten eingeben

000002E2 eingeben        clr.w seite          ;Zähler löschen
000002E8 weiein          move.w #160,d0
000002EC                 move.w seite,d1      ;Stringstart=
000002F2                 mulu d1,d0
000002F4                 addi.l #wertestr,d0  ;Tabellenanflag+160*Seite
000002FA                 move.l d0,a0
000002FC                 move.w #40,d0        ;Daten in Puffer kopieren
00000300                 lea puffer,a1
00000306 einloop         move.l (a0)+,(a1)+
00000308                 subq.w #1,d0
0000030A                 bne einloop
```

```
0000030E                 move.b seite+1,d0        ;Strings vorbereiten:
00000314                 addi.b #48,d0
00000318                 move.w #10,d1            ;Seitennummer + 48 (ASCII)
0000031C                 lea stringtabg,a0
00000322 einloop1        move.l (a0)+,a1          ;=Zehnerstelle
00000324                 move.b d0,(a1)
00000326                 subq.w #1,d1
00000328                 bne einloop1

0000032C                 lea einwindow,a0         ;Window für Eingabe
00000332                 bsr mainwindow           ;öffnen
00000336                 move.l iwindowpointer,a0
0000033C                 CALLINT CloseWindow      ;Fenster schließen
00000346                 move.w d2,a4             ;Gadget-ID retten
00000348                 move.w #160,d0
0000034C                 move.w seite,d1          ;Stringstart=
00000352                 mulu d1,d0
00000354                 addi.l #wertestr,d0      ;Tabellenanfag+160*Seite
0000035A                 move.l d0,a0
0000035C                 move.w #40,d0            ;Puffer zurückkopieren
00000360                 lea puffer,a1
00000366 einloop2        move.l (a1)+,(a0)+
00000368                 subq.w #1,d0
0000036A                 bne einloop2
0000036E                 move.w #40,d0
00000372                 move.w seite,d1          ;Fließkommastart=
00000378                 mulu d1,d0
0000037A                 addi.l #werte,d0         ;Tabellenanflag+40*Seite
00000380                 move.l d0,a3
00000382                 bsr wandel               ;umwandeln in Fließkomma
00000386                 move.w a4,d0
00000388                 cmpi.w #111,d0
0000038C                 bne endein               ;weitere Eingaben gewünscht
00000390                 move.w seite,d0
00000396                 addq #1,d0
00000398                 move.w d0,seite          ;schon 10 Seiten bearbeitet
0000039E                 cmpi.w #10,d0
000003A2                 bne weiein               ;nein, weitere Eingaben
000003A6 endein          bra exit                 ;fertig

000003AA seite         dc.w 0

000003AC stringtabg   dc.l string101,string111,string121,string131,string141
000003C0              dc.l string151,string161,string171,string181,string191
```

Auch bei der Eingabe der Datensätze wurde das Prinzip der Wertetabelle übernommen, der Vorteil ist hier aber noch eklatanter: Normalerweise müßte man für 100 Datensätze auch 100 Gadget-Strukturen anlegen, was einen gigantischen Quelltext- und Speicherplatzbedarf zur Folge hätte. Wir benutzen aber immer die gleichen zehn Gadgets, indem jeweils der Puffer der aktuell angewählten Gadgets in den Eingabepuffer kopiert werden. Da alle Puffer unmittelbar hintereinander liegen, errechnet sich bei einer Textpufferlänge von 16 Zeichen pro Gadget die Startadresse des Puffers zu Tabellenanfang +160*Seitennummer, die Seitennummer läuft von 0 bis 9 mit zehn Gadgets pro Seite. Dies alles ist nicht neu für Sie, jedoch das folgende Problem.

Mit unseren Gadgets müssen wir unbedingt die Nummer des Eingabefeldes ausgeben, um den Anwender nicht ziellos zwischen zehn völlig identisch aussehenden Eingabeseiten hin- und herblättern zu lassen. Leider ist der Zeiger in der Textstruktur aber unabhängig von der angeklickten Seite immer auf einen String gerichtet, wir müssen aber auf jeder Seite eine andere Nummer haben! Wie Sie sicher schon bemerkt haben, konnte auch dieses Problem gelöst werden. Dazu wurde vor dem Öffnen des jeweiligen Fensters die Zehnerstelle dieser Zahl modifiziert, indem zunächst die Seitennummer ausgelesen und dazu der Wert 48 (ASCII-Darstellung für Null) addiert wird. Das Resultat wird in das erste Zeichen der zehn Strings eingetragen, wobei die Stringanfänge in der Tabelle »stringtabg« abgelegt sind.

Im weiteren Verlauf der Routine wird in dem Ihnen schon bekannten Unterprogramm »mainwindow« das Eingabefenster geöffnet und auf eine Message gewartet. Nach Beendigung der Eingabe wird der Eingabepuffer der zehn Gadgets in den Pseudo-Puffer der entsprechenden Seite zurückkopiert und die Daten ins Fließkommaformat umgewandelt. Auch hierfür wird zunächst die Speicheradresse berechnet. Da jede FFP-Zahl 4 Byte belegt, ergibt sich der Puffer einer Seite zu Tabellenstart+40*Seitennummer. Werden weitere Eingaben gewünscht, wird nach der Umwandlung im Unterprogramm »wandel« zurück an den Anfang der Routine gesprungen, nachdem die Seitennummer um eins erhöht wurde. Falls aber die zehnte Seite erreicht ist, wird die Eingabe abgebrochen. Zum Schluß noch eine Bemerkung zum Gadget-Refresh: Dies ist bei jeder neu bearbeiteten Seite erforderlich, da wir sowohl den Gadget-Text als auch den Inhalt des Textpuffers geändert haben. Wir könnten eigentlich dazu die Funktion RefreshGadget heranziehen, ich habe aber extra einmal zu Demonstrationszwecken einen anderen Weg gewählt: Nach jeder beendeten Seite wird das Window geschlossen und mit der neuen Seite wieder geöffnet. Auch hierdurch werden nämlich alle Gadget-Daten aktualisiert!

4.6.4 Daten laden und speichern

HiSoft GenAmiga Assembler 1.21   "Chart-Darstellung"

```
          * Menüpunkt laden

000003D4 laden              lea diskwindow,a0        ;Window öffnen
000003DA                    bsr mainwindow
```

```
000003DE                    cmpi.w #111,d2          ;Abbruch
000003E2                    bne exit10
000003E6 ld1                move.l #1005,d2         ;Modus alt
000003EC                    move.l #puffer+160,d1   ;Zeiger auf Filenamen
000003F2                    CALLDOS Open            ;File öffnen
000003FC                    move.l d0,file          ;Pointer sichern
00000402                    beq fehler              ;Fehler!
00000406                    move.l d0,d1
00000408                    move.l #wertestr,d2     ;Zeiger auf Daten
0000040E                    move.l #2000,d3         ;Datei lesen
00000414                    CALLDOS Read
0000041E                    bra Schliess            ;Datei schließen
00000422 fehler             bsr requester           ;Fehler abfragen
00000426                    tst.l d0
00000428                    bne ld1                 ;nochmal versuchen
0000042C                    bra exit10


         * Menüpunkt speichern

00000430 speichern          lea diskwindow,a0       ;Window öffnen
00000436                    bsr mainwindow
0000043A                    cmpi.w #111,d2          ;Abbruch
0000043E                    bne exit10
00000442 sp1                move.l #1006,d2         ;Modus neu
00000448                    move.l #puffer+160,d1   ;Zeiger auf Filenamen
0000044E                    CALLDOS Open            ;File öffnen
00000458                    move.l d0,file          ;Pointer sichern
0000045E                    beq fehler1             ;Fehler!
00000462                    move.l d0,d1
00000464                    move.l #wertestr,d2     ;Zeiger auf Daten
0000046A                    move.l #2000,d3         ;2000 Byte schreiben
00000470                    CALLDOS Write
0000047A Schliess           move.l file,d1          ;Pointer laden
00000480                    CALLDOS Close           ;Datei schließen
0000048A                    bra exit10
0000048E fehler1            bsr requester           ;Fehler abfragen
00000492                    tst.l d0
00000494                    bne sp1                 ;nochmal versuchen
00000498 exit10             move.l iwindowpointer,a0
0000049E                    CALLINT CloseWindow     ;Fenster schließen
000004A8                    bra exit

000004AC file               dc.l 0                  ;File-Pointer

         * ZWEITES MENÜ (OPTION)

         * Menüpunkt Ende

000004B0 endof              bra ende
```

Da die DOS-Funktionen noch nicht behandelt wurden, möchte ich Sie an dieser Stelle nicht mit neuen Begriffen und Funktionen überfallen. Die Arbeitsweise der beiden Routinen ist aber so einfach, daß Sie sie nach der Lektüre des folgenden Kapitels mit Sicherheit nachvollziehen können.

4.6.5 Darstellung der Tortengrafik

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * Darstellung Tortengrafik

000004B4 torte           lea werte,a0
000004BA                 move.l (a0),d0            ;Keine Werte vorhanden?
000004BC                 beq wait11
000004C0                 bsr plot                  ;Bildschirm aufbauen
000004C4                 bsr reserve               ;Platz reservieren

         * Torte mit "normalen" Grafikbefehlen aufbauen

000004C8                 move.l iwindowpointer,a0
000004CE                 move.l 50(a0),d7          ;Rast-Port holen
000004D2                 clr.l d5                  ;Startwinkel=PI Halbe
000004D4                 move.l PI,d6              ;Endwinkel = 3 * PI Halbe
000004DA                 move.l ym,d0              ;Mittelpunkt y
000004E0                 add.l off,d0
000004E6                 bsr ellipse               ;Ellipsenteil zeichnen
000004EA                 clr.l d5                  ;Startwinkel
000004EC                 move.l PIm2,d6            ;Endwinkel = 2 * PI
000004F2                 move.l ym,d0              ;Mittelpunkt y
000004F8                 bsr ellipse               ;Ellipse zeichnen
000004FC                 move.l d7,a1              ;Rastport holen
000004FE                 move.l xm,d0
00000504                 sub.l xr,d0
0000050A                 move.l ym,d1
00000510                 CALLGRAF Move             ;Verbindungslinie
0000051A                 move.l d7,a1
0000051C                 move.l xm,d0              ;linke Seite zeichnen
00000522                 sub.l xr,d0
00000528                 move.l ym,d1
0000052E                 add.l off,d1
00000534                 CALLGRAF Draw
0000053E                 move.l d7,a1              ;Rastport holen
00000540                 move.l xm,d0
00000546                 add.l xr,d0
0000054C                 move.l ym,d1
00000552                 CALLGRAF Move             ;Verbindungslinie
```

```
0000055C                move.l d7,a1
0000055E                move.l xm,d0            ;rechte Seite zeichnen
00000564                add.l xr,d0
0000056A                move.l ym,d1
00000570                add.l off,d1
00000576                CALLGRAF Draw
00000580                bsr vor                 ;Ausgabe vorbereiten
00000584                move.l d2,d6            ;Zähler für Summenbildung
00000586                move.l d4,a5
00000588                clr.l d0                ;Summe löschen
0000058A summe          move.l (a5)+,d1         ;Nächster Wert
0000058C                CALLFFP SPAdd           ;hinzuaddieren
00000596                subq #1,d6              ;Zähler abgelaufen?
00000598                bne summe
0000059C                move.l d0,d5

         * Tortenstücke malen

0000059E                clr.l altx
000005A4                clr.l alty
000005AA                move.l xrfloat,d0       ;Faktoren durch zwei
000005B0                move.l zwei,d1
000005B6                CALLFFP SPDiv           ;vorbereiten
000005C0                move.l d0,xrfloatd2
000005C6                move.l yrfloat,d0
000005CC                move.l zwei,d1
000005D2                CALLFFP SPDiv
000005DC                move.l d0,yrfloatd2
000005E2                move.l PIm2,d0          ;2*PI/Summe=Schrittweite
000005E8                move.l d5,d1            ;des Winkels
000005EA                CALLFFP SPDiv
000005F4                move.l d0,d5
000005F6                move.l d7,a1            ;Rast-Port holen
000005F8                move.l xm,d0
000005FE                move.l ym,d1            ;Startpunkt setzen
00000604                CALLGRAF Move
0000060E                move.l d7,a1            ;Rast-Port holen
00000610                move.l xm,d0            ;Linie zum
00000616                add.l xr,d0
0000061C                move.l ym,d1            ;Tortenrand
00000622                CALLGRAF Draw           ;zeichnen
0000062C                clr.l d6                ;aktueller Winkel
0000062E                move.b #1,flag          ;Flag setzen
00000636                move.l d4,a3
00000638 torteloop      move.l (a3)+,d0         ;Wert holen
0000063A                move.l d5,d1            ;mit Winkel-Schrittweite
0000063C                CALLFFP SPMul           ;multiplizieren
```

```
00000646            move.l d6,d1                ;Zu Startwinkel addieren
00000648            CALLFFP SPAdd
00000652            move.l d0,d3                ;und merken
00000654            CALLMATHTRANS SPCos         ;Cosinus bilden
0000065E            move.l xrfloat,d1
00000664            CALLFFP SPMul               ;mal x-Radius=Offset
0000066E            move.l xmfloat,d1           ;plus x-Mittelpunkt=
00000674            CALLFFP SPAdd               ;Koordinate
0000067E            CALLFFP SPFix               ;Integer
00000688            move.l d0,a4                ;merken
0000068A            move.l d3,d0                ;Winkel wiederholen
0000068C            CALLMATHTRANS SPSin         ;Sinus bilden
00000696            move.l yrfloat,d1
0000069C            CALLFFP SPMul               ;mal y-Radius=Offset
000006A6            move.l ymfloat,d1           ;plus y-Mittelpunkt=
000006AC            CALLFFP SPAdd               ;Koordinate
000006B6            CALLFFP SPFix               ;Integer
000006C0            move.l d0,a5                ;merken
000006C2            move.l d7,a1                ;Rast-Port holen
000006C4            move.l xm,d0
000006CA            move.l ym,d1                ;Startpunkt setzen
000006D0            CALLGRAF Move
000006DA            move.l d7,a1                ;Rast-Port holen
000006DC            move.l a4,d0                ;Linie zum
000006DE            move.l a5,d1                ;Tortenrand
000006E0            CALLGRAF Draw               ;zeichnen
000006EA            tst.b flag                  ;jedes zweite Stück
000006F0            bne nofill                  ;ausfüllen

        * Tortenstück füllen

000006F4            move.l d2,a2                ;d2 retten
000006F6            move.l d3,d0
000006F8            move.l d6,d1                ;Winkel Tortenstück
000006FA            CALLFFP SPSub
00000704            move.l zwei,d1              ;durch 2 teilen
0000070A            CALLFFP SPDiv
00000714            move.l d6,d1                ;zu altem Winkel
00000716            CALLFFP SPAdd               ;addieren
00000720            move.l d0,winkel            ;und merken
00000726            CALLMATHTRANS SPCos         ;Cosinus bilden
00000730            move.l xrfloatd2,d1
00000736            CALLFFP SPMul               ;mal x-Radius/2=Offset
00000740            move.l xmfloat,d1           ;plus x-Mittelpunkt=
00000746            CALLFFP SPAdd               ;Koordinate
00000750            CALLFFP SPFix               ;Integer
0000075A            move.l d0,xmerk             ;merken
```

```
00000760                    move.l winkel,d0         ;Winkel wiederholen
00000766                    CALLMATHTRANS SPSin      ;Sinus bilden
00000770                    move.l yrfloatd2,d1
00000776                    CALLFFP SPMul            ;mal y-Radius/2=Offset
00000780                    move.l ymfloat,d1        ;plus y-Mittelpunkt=
00000786                    CALLFFP SPAdd            ;Koordinate
00000790                    CALLFFP SPFix            ;Integer
0000079A                    move.l d0,ymerk          ;merken
000007A0                    move.l d7,a1             ;Rastport holen
000007A2                    moveq.l #1,d2            ;Füllmodus
000007A4                    move.l xmerk,d0
000007AA                    move.l ymerk,d1          ;Koordinaten
000007B0                    CALLGRAF Flood           ;Tortenstück füllen
000007BA                    move.l a2,d2             ;d2 wiederholen

000007BC nofill             move.l d3,d0
000007BE                    move.l PI,d1             ;Vorderansicht?
000007C4                    CALLFFP SPCmp
000007CE                    cmpi.l #-1,d0
000007D4                    beq no3D1                ;nein, keine
000007D8                    move.l d7,a1
000007DA                    move.l a4,d0             ;ja, Linie auf
000007DC                    move.l a5,d1
000007DE                    add.l off,d1             ;Vorderseite zeichnen
000007E4                    CALLGRAF Draw

000007EE no3D1              tst.b flag
000007F4                    bne no3D                 ;nur jedes zweite Stück
000007F8                    move.l alty,d0           ;PI überschritten?
000007FE                    cmpi.l #100,d0
00000804                    bls no3D                 ;ja, nicht füllen

         * Vorderansicht füllen

00000808                    move.l d2,a2             ;d2 retten
0000080A                    move.l altx,d0
00000810                    subq.l #1,d0             ;Startpunkt fast in Ecke
00000812                    move.l alty,d1           ;(x-1, y+5)
00000818                    addi.l #15,d1
0000081E                    move.l d7,a1             ;Rastport holen
00000820                    moveq.l #1,d2            ;Füllmodus setzen
00000822                    CALLGRAF Flood
0000082C                    move.l a2,d2             ;d2 wiederholen

0000082E no3D               move.l a4,altx           ;Werte merken
00000834                    move.l a5,alty
0000083A                    move.l d3,d6             ;Winkel neu setzen
```

```
0000083C                eor.b #1,flag
00000844                subq #1,d2
00000846                bne torteloop           ;nächstes Stück zeichnen

0000084A                bra mainpl              ;Fertig

0000084E mempoint       dc.l 0
00000852 PId2           dc.b $C9,$0F,$DA,$41    ;PI durch 2
00000856 PI             dc.b $C9,$0F,$DA,$42    ;PI
0000085A PIm3d2         dc.b $96,$CB,$E5,$43    ;PI mal 3 durch 2
0000085E PIm2           dc.b $C9,$0F,$DA,$43    ;PI mal 2
00000862 VIERH          dc.b $C8,$00,$00,$49    ;400 (Durchmesser Ellipse)
00000866 zwei           dc.b $80,$00,$00,$42    ;Zwei (Teilfaktor)

0000086A xm             dc.l 320                ;Mittelpunkt x
0000086E ym             dc.l 100                ;Mittelpunkt y
00000872 xr             dc.l 160                ;x-Radius
00000876 yr             dc.l 56                 ;y-Radius
0000087A off            dc.l 25                 ;3-D-Ellipsenversatz

0000087E xmfloat        dc.l 0
00000882 ymfloat        dc.l 0
00000886 xrfloat        dc.l 0
0000088A yrfloat        dc.l 0
0000088E xrfloatd2      dc.l 0
00000892 yrfloatd2      dc.l 0
00000896 xmerk          dc.l 0
0000089A ymerk          dc.l 0
0000089E winkel         dc.l 0
000008A2 altx           dc.l 0
000008A6 alty           dc.l 0


         * Ellipse(nabschnitt) zeichnen

000008AA ellipse        CALLFFP SPFlt           ;ym in Fließkomma
000008B4                move.l d0,ymfloat
000008BA                move.l xm,d0            ;xm in Fließkomma
000008C0                CALLFFP SPFlt
000008CA                move.l d0,xmfloat
000008D0                move.l xr,d0            ;xr in Fließkomma
000008D6                CALLFFP SPFlt
000008E0                move.l d0,xrfloat
000008E6                move.l yr,d0            ;yr in Fließkomma
000008EC                CALLFFP SPFlt
000008F6                move.l d0,yrfloat
```

```
000008FC          move.l VIERH,d1        ;Schrittweite berechnen
00000902          move.l PIm2,d0
00000908          CALLFFP SPDiv
00000912          move.l d0,schritt
00000918          move.l d5,d3           ;Startwinkel setzen
0000091A          move.b #1,flag         ;Flag setzen
00000922 elloop   move.l d3,d0
00000924          CALLMATHTRANS SPCos
0000092E          move.l xrfloat,d1
00000934          CALLFFP SPMul          ;x-Offset=xr*cos(w)
0000093E          move.l xmfloat,d1
00000944          CALLFFP SPAdd          ;Punkt=Mittelpunkt+Offset
0000094E          CALLFFP SPFix          ;Integer-Wert
00000958          move.l d0,d2           ;merken
0000095A          move.l d3,d0           ;Winkel wiederholen
0000095C          CALLMATHTRANS SPSin
00000966          move.l yrfloat,d1
0000096C          CALLFFP SPMul          ;y-Offset=yr*sin(w)
00000976          move.l ymfloat,d1
0000097C          CALLFFP SPAdd          ;Punkt=Mittelpunkt+Offset
00000986          CALLFFP SPFix          ;Integer-Wert
00000990          move.l d0,d1           ;merken
00000992          move.l d2,d0           ;x-Wert holen
00000994          move.l d7,a1           ;Rast-Port holen
00000996          tst.b flag             ;Erstes Mal: Punkt setzen
0000099C          bne Pixel
000009A0          CALLGRAF Draw          ;Sonst: Linie
000009AA          bra Linie
000009AE Pixel    CALLGRAF Move
000009B8 Linie    clr.b flag             ;Flag setzen
000009BE          move.l d3,d1
000009C0          move.l schritt,d0
000009C6          CALLFFP SPAdd          ;Winkel erhöhen
000009D0          move.l d0,d3           ;und neu setzen
000009D2          move.l d6,d1           ;mit Endwinkel
000009D4          CALLFFP SPCmp          ;vergleichen
000009DE          cmpi.l #-1,d0
000009E4          bne elloop             ;noch nicht erreicht
000009E8          rts

000009EA schritt  dc.l 0
000009EE flag     dc.b 0
                  even
```

Nachdem als erstes überprüft wurde, ob der Anwender überhaupt irgendwelche Werte eingegeben hat, kann es losgehen: Der Aufbau einer Tortengrafik vollzieht sich in fünf Abschnitten:

- Aufbau der Tortenränder (2 Ellipsen für 3-dimensionale Darstellung)
- Zeichnen der Tortenstückbegrenzungen auf der Torte
- Ausfüllen jedes zweiten Tortenstückes
- Zeichnen der Tortenstückbegrenzungen auf der Vorderansicht der Torte
- Ausfüllen jeder zweiten Tortenstückvorderansicht

An Hand von Bild 4.3 können Sie sich diese Vorgehensweise noch einmal verdeutlichen.

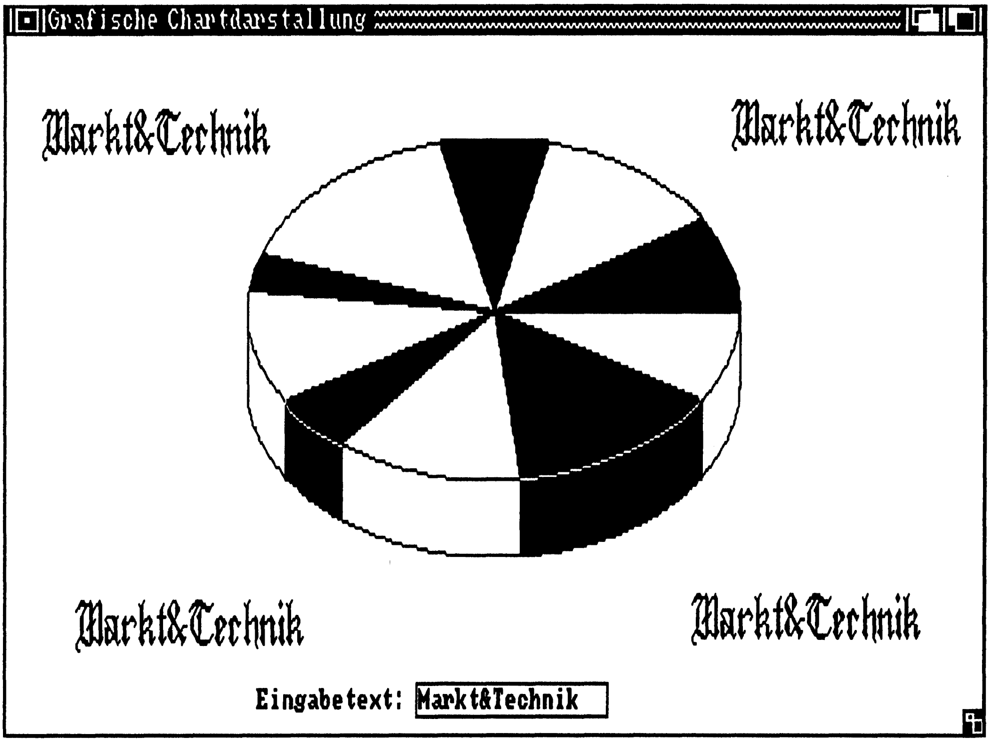


Bild 4.3: Tortengrafik

Als erstes muß die Torte aufgebaut werden. Dazu ist eine Voll- und eine Teilellipse für den unteren Tortenrand der Vorderansicht erforderlich. Da die DrawEllipse-Routine versagt, mußte ein anderer Weg gefunden werden. Eine Ellipse kann man sich einfach als Punktmenge denken, die nachfolgende Gleichungen erfüllen:

x = XM + XR*COS(W) und y = YM + YR*SIN(W)

mit XM, YM : Koordinaten des Ellipsenmittelpunktes
XR, YR : Radien in x- bzw. y-Richtung
W : Winkel (läuft von 0 bis 360 Grad bzw. im Bogenmaß von 0 bis 2*PI)

Im nächsten Abschnitt werden wir uns ausführlich mit der Darstellung der Ellipse beschäftigen, vielleicht blättern Sie ja schon einmal ein paar Seiten vor! Um also die Koordinaten der einzelnen Punkte der Ellipse zu erhalten, muß man nur den Winkel von 0 bis 360 Grad laufen lassen, jeweils die Winkelfunktionen zu diesem Winkel bestimmen und dieses in die Gleichungen einsetzen. Die Genauigkeit der gezeichneten Ellipse hängt davon ab, in welcher Schrittweite man den Winkel erhöht. Natürlich sind möglichst kleine Schritte angebracht, um eine große Genauigkeit zu erzielen, damit steigt aber auch die benötigte Rechenzeit an. Im allgemeinen bietet sich die Schrittweite an, die man aus der Division des Ellipsendurchmessers durch 360 (2*PI im Bogenmaß) erhält, wobei der maximal auftretende Durchmesser einzusetzen ist, bei unserer Ellipse der in x-Richtung.

In dem Unterprogramm »Ellipse« wird das beschriebene Verfahren durchgezogen: Zu beachten ist dabei nur noch, daß je zwei Ellipsenpunkte durch eine Linie verbunden werden müssen, da unsere Genauigkeit nicht so groß ist, daß zwei Punkte genau nebeneinander liegen. Sie werden feststellen, liebe Leser, daß die Zeichengeschwindigkeit alles andere als hoch ist, da besonders die Berechnung der Winkelfunktionen sehr langsam ist. Deshalb werden wir uns im nächsten Abschnitt auch eines verbesserten, aber auch aufwendigeren Verfahrens zuwenden.

Um unsere Torte aufzubauen, wird die Ellipsenroutine zweimal aufgerufen. Anschließend werden der untere Tortenrand und die Draufsicht noch durch zwei Linien verbunden, womit die 3-D-Torte auch schon fertig ist.

Als nächstes ist die Einteilung der Torte in die einzelnen Tortenstücke vorzunehmen. Hierfür gilt, daß der Winkel eines Tortenstücks sich zu 360 Grad verhält, wie der durch das Tortenstück repräsentierte Wert zu der Summe aller Werte. Im Unterschied zu Kurven- und Säulengrafiken wird durch ein solches Diagramm nämlich kein Absolutwert dargestellt, sondern der Anteil bestimmter Daten an einer Gesamtmenge (z.B. Sitze einer bestimmten Partei im Verhältnis zu allen, zur Verfügung stehenden Sitzen). Deshalb hängt die Größe eines Tortenstückes auch nicht nur wie z.B. die Höhe einer Säule von dem dargestellten Wert, sondern vor allen Dingen auch von der Anzahl der betrachteten Werte ab, wenn ich nur drei Tortenstücke aufbaue, sind die einzelnen natürlich viel größer, als wenn ich zehn darstelle, obwohl sich an den einzelnen Werten ja nichts geändert hat!

Daher bilden wir zunächst die Summe aller Werte und legen dann die Winkelschrittweite als 2*PI/Summe fest. Den zu einem bestimmten Wert gehörenden Winkel erhalten wir nun dadurch, daß wir diesen Wert mit der Winkelschrittweite multiplizieren. Jetzt können wir ab dem Label »torteloop« die Linien vom Ellipsenmittelpunkt zum Rand ziehen, indem wir den berechneten Winkel wieder in unsere Ellipsenformeln einsetzen und damit den Punkt auf dem Ellipsenrand erhalten.

Es folgt der Programmabschnitt, mit dem jedes zweite Tortenstück ausgefüllt wird. Dazu existiert ein Flag, das in der Schleife wechselweise gesetzt und gelöscht wird und als Kennzeichnung für eine erforderliche Füllung dient. Um eine Fläche »sicher« ausfüllen zu können, d.h., auch bei noch so kleinen Abständen der Ränder zueinander ein Überfließen der Farbe auf den restlichen Bildschirm zu verhindern, muß man die Füllfunktion am Mittelpunkt der Fläche starten. Natürlich hat ein Tortenstück keinen »echten« Mittelpunkt, wenn man aber den überstrichenen Winkel und den Radius jeweils halbiert, bekommt man zumindest für unsere Zwecke den Punkt, der ein sicheres Füllen auch dann garantiert, wenn es sich um ein sehr schmales Tortenstück handelt:

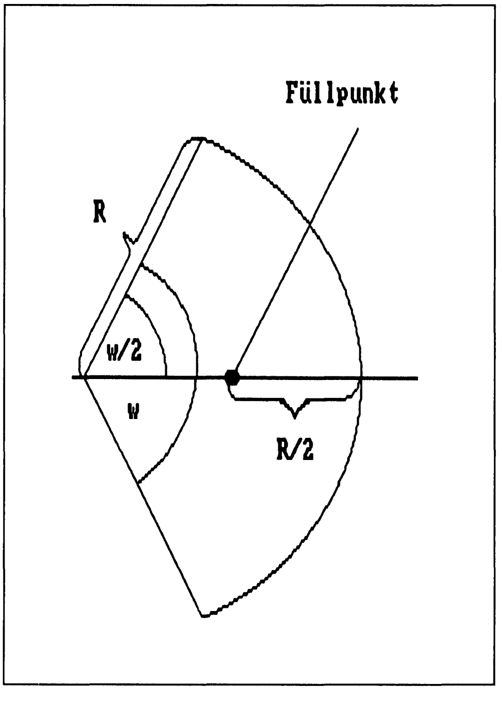


Bild 4.4: Sicheres Füllen eines Tortenstücks mit Farbe

Nun können wir uns der Betrachtung der Vorderansicht zuwenden: Diese ist nur dann relevant, wenn sich das Tortenstück im Bereich von 0 bis PI bewegt, da man ja nur in diesem Bereich einen Blick auf den Tortenrand genießen kann. Auch hier wird wieder als erstes eine Linie gezogen, diesmal vom oberen zum unteren Tortenrand. Da wir oben die Koordinaten des Punktes am oberen Tortenrand schon berechnet hatten, um die Linie vom Ellipsenmittelpunkt dorthin zu zeichnen, können wir einfach diesen Punkt als Ausgang nehmen und um den y-Offset, um den die beiden Ränder verschoben sind, senkrecht nach unten zeichnen.

Das größte Problem besteht in der Füllung des Tortenrandes: Es kann nämlich vorkommen, daß ein Tortenstück bei einem Winkel beginnt, der kleiner als 180 Grad (PI) ist, aber bei einem endet, der jenseits dieser Grenze liegt. Die Folge ist, daß nur ein Teil des Torten-

stückes perspektivisch zu sehen ist, weshalb die Methode, die wir oben benutzt haben, auch nicht funktionieren kann, wie Bild 4.5 zeigt.

Man müßte eigentlich zunächst den Anteil des Winkels berechnen, der bis zur Grenze PI überstrichen wurde, diesen halbieren und damit schließlich den Startpunkt zum Füllen berechnen. Wir können es uns aber auch einfacher machen, indem wir einfach den Startpunkt direkt neben die Verbindungslinie vom oberen und unteren Tortenrand legen, d.h., wir wählen den kleinstmöglichen Abstand von einem Punkt in x-Richtung, wodurch ein sicheres Füllen ebenfalls garantiert ist (Bild 4.6).

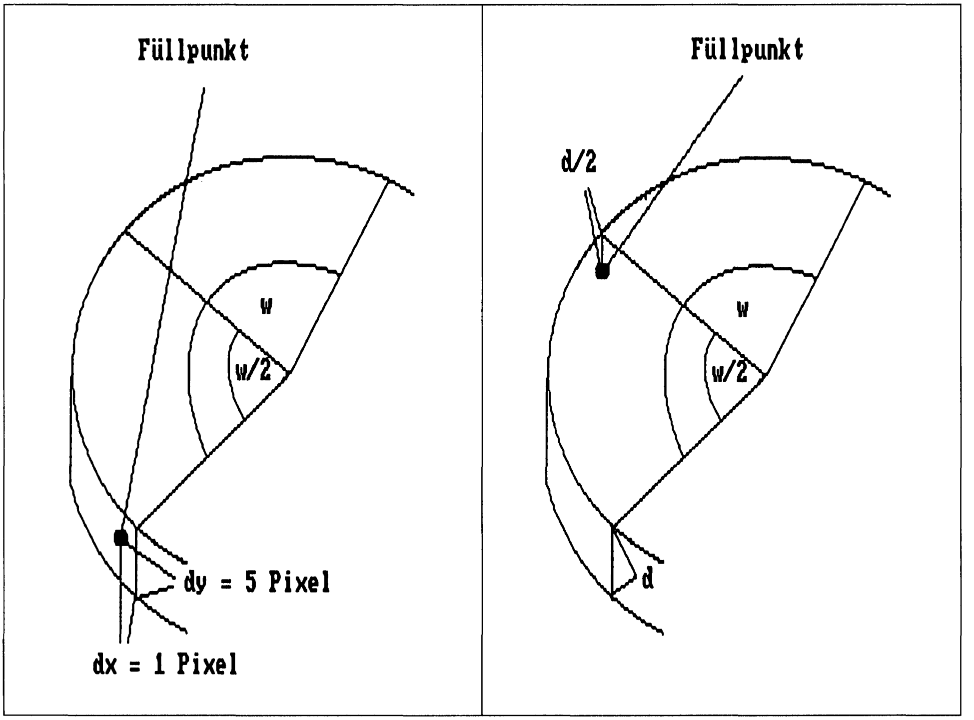


Bild 4.5: Fehlerhafte Lage des Füllpunkts für den Tortenrand

Bild 4.6: Korrekte Lage des Füllpunkts für den Tortenrand

Damit ist ein Tortenstück komplett behandelt worden, das nächste Stück ist an der Reihe. Obwohl der Programmieraufwand für eine 3-D-Torte relativ hoch ist, ziehe ich diese Darstellung einer lieblos hingemalten Kreisscheibe mit ausschließlich leeren Feldern vor, wie man sie sogar in kommerziell angebotenen Programmen findet. Aber auch hier können Sie noch eigene Verbesserungen einfügen, wie z.B. die Änderung des Musters bei jedem Tortenstück, u.s.w. Der Phantasie sind keine Grenzen gesetzt!

4.6.6 Darstellung der Säulengrafik

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * Darstellung Säulengrafik

000009F0 saeule          lea werte,a0
000009F6                 move.l (a0),d0          ;Keine Werte vorhanden?
000009F8                 beq wait11
000009FC                 bsr plot                ;Bildschirm aufbauen
00000A00                 bsr reserve             ;Platz reservieren
         * Feld für Säulen aufbauen und Maßstab ausrechnen

00000A04                 move.l iwindowpointer,a0
00000A0A                 move.l 50(a0),a1        ;Zeiger auf Rastport
00000A0E                 move.l a1,d7            ;und merken
00000A10                 move.l #94,d0           ;Startpunkt setzen
00000A16                 move.l #27,d1
00000A1C                 CALLGRAF Move           ;(Grafikcursor)
00000A26                 move.l d7,a1            ;Rastport wiederholen
00000A28                 lea koord,a0            ;Koordinatenangaben
00000A2E                 move.l #23,d0           ;23 Koordinatenpaare
00000A34                 CALLGRAF PolyDraw       ;Polygon zeichnen
00000A3E                 bsr vor                 ;Daten vorbereiten
00000A42                 move.l d2,d6            ;Zähler setzen
00000A44                 move.l d4,a5
00000A46                 clr.l maximum           ;Maximum löschen
00000A4C maxloop         move.l (a5)+,d0         ;Wert holen
00000A4E                 move.l maximum,d1       ;mit Maximum
00000A54                 CALLFFP SPCmp           ;vergleichen
00000A5E                 cmpi.l #-1,d0           ;größer
00000A64                 bne nomax               ;nein
00000A68                 move.l -(a5),maximum    ;ja, neues Maximum
00000A6E                 move.l (a5)+,d0         ;Zeiger korrigieren
00000A70 nomax           subq.b #1,d6
00000A72                 bne maxloop             ;Weitere Werte prüfen
00000A76                 move.l einhuvier,d0     ;140 in Fließkomma
00000A7C                 move.l maximum,d1       ;Maßstab=140 Punkte
00000A82                 CALLFFP SPDiv           ;durch maximalen Wert
00000A8C                 move.l d0,höhenmass     ;für Höhe Säulen
00000A92                 move.l d2,d0            ;Anzahl der Werte
00000A94                 CALLFFP SPFlt           ;in Fließkomma
00000A9E                 move.l d0,d1
00000AA0                 move.l vierhufünf,d0    ;450 in Fließkomma
00000AA6                 CALLFFP SPDiv           ;durch Anzahl = DX
00000AB0                 move.l d0,d6            ;merken
00000AB2                 CALLFFP SPFix           ;Integer
```

```
00000ABC                  move.l d0,gesamtbreit
00000AC2                  move.l d6,d0
00000AC4                  move.l nullsieben,d1     ;0.75 in Fließkomma (70%)
00000ACA                  CALLFFP SPMul            ;=Säulenbreite
00000AD4                  CALLFFP SPFix
00000ADE                  move.l d0,säulenbreit

         * Einzelne Säulen aufbauen

00000AE4                  move.l #50,xstart        ;linker Eckpunkt von Säule
00000AEE                  move.l d7,a0             ;Rastport holen
00000AF0                  move.l #areainf,$10(a0) ;Area Pointer in Rastport
00000AF8                  move.l d4,a5             ;Zeiger auf Werte
00000AFA                  clr.b flag               ;Flag löschen
00000B00 säulenloop       move.l d7,a1             ;Rastport holen
00000B02                  moveq.l #1,d0            ;Farbe setzen für Front
00000B04                  CALLGRAF SetAPen
00000B0E                  move.l (a5)+,d0          ;Wert holen
00000B10                  move.l höhenmass,d1      ;mal Maßstab
00000B16                  CALLFFP SPMul            ;=Höhe in Punkten
00000B20                  CALLFFP SPFix            ;natürlich Integer-Wert
00000B2A                  move.l d0,höhe           ;merken
00000B30                  move.l d2,a2             ;d2 retten
00000B32                  move.l xstart,d0
00000B38                  move.l #185,d3
00000B3E                  move.l d3,d1
00000B40                  sub.l höhe,d1            ;oberer Punkt links
00000B46                  move.l d1,d4             ;merken
00000B48                  move.l d0,d2             ;unterer Punkt rechts
00000B4A                  add.l säulenbreit,d2
00000B50                  move.l d2,d5             ;merken
00000B52                  move.l d7,a1             ;Rastport holen
00000B54                  CALLGRAF RectFill        ;Vorderfront zeichnen
00000B5E                  move.l d7,a1             ;Rastport holen
00000B60                  moveq.l #3,d0            ;Farbe setzen für
00000B62                  CALLGRAF SetAPen         ;restliche Seiten
00000B6C                  move.l xstart,d0
00000B72                  move.l d4,d1
00000B74                  move.l d7,a1
00000B76                  CALLGRAF AreaMove        ;Erster Eintrag
00000B80                  move.l xstart,d0
00000B86                  addi.l #54,d0            ;x plus 45
00000B8C                  move.l d0,d6             ;merken
00000B8E                  move.l d4,d1             ;y holen
00000B90                  subi.l #18,d1            ;minus 15
00000B96                  move.l d1,a4             ;merken
00000B98                  move.l d7,a1
```

```
00000B9A              CALLGRAF AreaDraw     ;zweiter Eintrag
00000BA4              move.l d6,d0          ;x plus
00000BA6              add.l säulenbreit,d0  ;Säulenbreite
00000BAC              move.l d0,d6          ;merken
00000BAE              move.l a4,d1          ;y bleibt gleich
00000BB0              move.l d7,a1
00000BB2              CALLGRAF AreaDraw     ;dritter Eintrag
00000BBC              move.l d6,d0          ;x bleibt gleich
00000BBE              move.l a4,d1
00000BC0              add.l höhe,d1         ;y plus Höhe
00000BC6              move.l d7,a1
00000BC8              CALLGRAF AreaDraw     ;vierter Eintrag
00000BD2              move.l d5,d0          ;altes x holen
00000BD4              move.l #185,d1        ;altes y holen
00000BDA              move.l d7,a1          ;Rastport holen
00000BDC              CALLGRAF AreaDraw     ;fünfter Eintrag
00000BE6              move.l d5,d0          ;x bleibt
00000BE8              move.l d4,d1          ;195 minus Höhe
00000BEA              move.l d7,a1          ;Rastport holen
00000BEC              CALLGRAF AreaDraw     ;sechster Eintrag
00000BF6              move.l xstart,d0      ;Start x
00000BFC              move.l d4,d1          ;195 minus Höhe
00000BFE              move.l d7,a1          ;Rastport holen
00000C00              CALLGRAF AreaDraw     ;siebter Eintrag
00000C0A              move.l d7,a1
00000C0C              CALLGRAF AreaEnd      ;Polygon schließen+füllen
00000C16              move.l xstart,d0
00000C1C              add.l gesamtbreit,d0  ;Startwert für nächste
00000C22              move.l d0,xstart      ;Säule setzen
00000C28              move.l a2,d2          ;d2 wiederholen
00000C2A              subq.w #1,d2
00000C2C              bne säulenloop
00000C30              move.l d7,a1          ;Rastport holen
00000C32              clr.l d0              ;alte Farbe setzen
00000C34              CALLGRAF SetAPen
00000C3E              bra mainpl            ;fertig


00000C42 koord        dc.w 94,27,40,45,40,185,94,167,94,27,547,27
00000C5A              dc.w 547,167,94,167,40,185,493,185,547,167,547,132
00000C72              dc.w 94,132,40,150,40,115,94,97,547,97,547,62
00000C8A              dc.w 94,62,40,80,40,45,94,27,94,167

00000C9E vierhufünf   dc.b $E1,$00,$00,$49  ;450 in Fließkomma
00000CA2 einhuvier    dc.b $8C,$00,$00,$48  ;140 in Fließkomma
00000CA6 nullsieben   dc.b $C0,$00,$00,$40  ;0.75 in Fließkomma
```

```
00000CAA säulenbreit      dc.l 0
00000CAE gesamtbreit      dc.l 0
00000CB2 höhenmass        dc.l 0
00000CB6 maximum          dc.l 0
00000CBA xstart           dc.l 0
00000CBE höhe             dc.l 0

00000CC2 platz            ds.l 20

         * Reserviert Platz für Area-Befehle und Flood

00000D12 reserve          move.l iwindowpointer,a0
00000D18                  move.l 50(a0),a0          ;Rast-Port holen
00000D1C                  move.l #tmpras,$0C(a0)    ;Strukur eintragen
00000D24                  rts

         * Plotschirm aufbauen

00000D26 plot     lea plot_defs,a0                  ;Screen öffnen
00000D2C          CALLINT OpenScreen
00000D36          move.l d0,plotpointer             ;Pointer für
00000D3C          move.l d0,point4
00000D42          lea plotwindow,a0
00000D48          CALLINT OpenWindow                ;Window öffnen
00000D52          move.l d0,iwindowpointer
00000D58          rts

         * Ausgabe vorbereiten

00000D5A vor      lea strinfo13,a0                  ;Startsatz holen
00000D60          move.l 28(a0),d3                  ;aus Stringinfo
00000D64          lea strinfo14,a0                  ;Zielsatz holen
00000D6A          move.l 28(a0),d2                  ;aus Stringinfo
00000D6E          move.l d3,d4                      ;Startadresse erster
00000D70          lsl.l #2,d4                       ;Datensatz berechnen
00000D72          addi.l #werte,d4                  ;(*4 wegen Langwörtern)
00000D78          sub.l d3,d2                       ;Anzahl Datensätze
00000D7A          addq.l #1,d2                      ;berechnen
00000D7C          rts
```

Die Darstellung der Säulengrafik ist wesentlicher einfacher zu verstehen als die des Tortendiagramms. Nachdem als erstes wiederum geprüft wurde, ob irgendwelche Werte eingegeben wurden, wird der Grafikscreen mit einem Window geöffnet. Als erstes wird dann ein 3-D-Rahmen gezeichnet, in den die Säulen perspektivisch eingebunden werden (Bild 4.7).

Um den Maßstab für die Höhe der Säulen zu berechnen, muß das Maximum der darzustellenden Werte gefunden werden. Das Verfahren dazu ist schon von dem Wertetabelle-Programm bekannt: Der erste Wert wird als Maximum definiert. Alle anderen Werte werden dann mit diesem Maximum verglichen und dieses gegebenenfalls durch einen höheren Wert ersetzt. Da wir in y-Richtung 140 Punkte zur Verfügung haben, ergibt sich die Schrittweite zu 140/Maximum. In x-Richtung stehen 450 Punkte zur Verfügung. Damit ergibt sich für jede Säule ein Abschnitt von (450/Anzahl Werte) Punkten. Da wir aber zwischen den einzelnen Säulen aber noch eine Lücke benötigen, können wir diesen Abschnitt nicht voll ausnutzen, sondern müssen ihn noch mit einem Wert, der kleiner als 1 ist, multiplizieren. Als recht ansehnlich für das Verhältnis von Säulenbreite zu Lückenbreite hat sich der Wert 3 herausgestellt, d.h. während die Säule drei Viertel des zur Verfügung stehenden Platzes einnimmt, bekommt die Lücke davon nur ein Viertel.

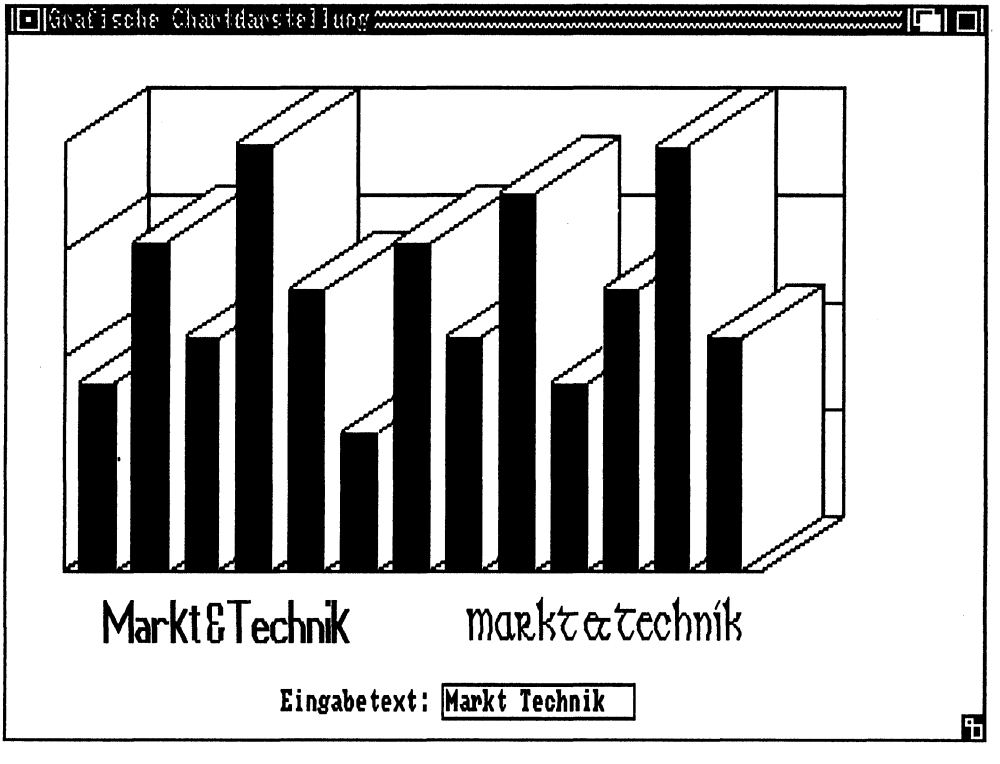


Bild 4.7: Säulengrafik

Jetzt wird die Vorderfornt mittels der RectFill-Funktion gezeichnet, indem zum Startpunkt in der linken, Ecke die Säulenbreite und -höhe hinzugerechnet werden. Die durch die perspektivische Verzerrung zu sehenden Seiten werden anschließend mit den Area-Funktionen blitzschnell aufgebaut, wozu die sieben erforderlichen Eckpunkte in die Area-Liste

eingetragen werden (Bild 4.8). Dies geschieht natürlich in einer zur Vorderseite unterschiedlichen Farbe, um den 3-D-Effekt noch mehr hervorzuheben.

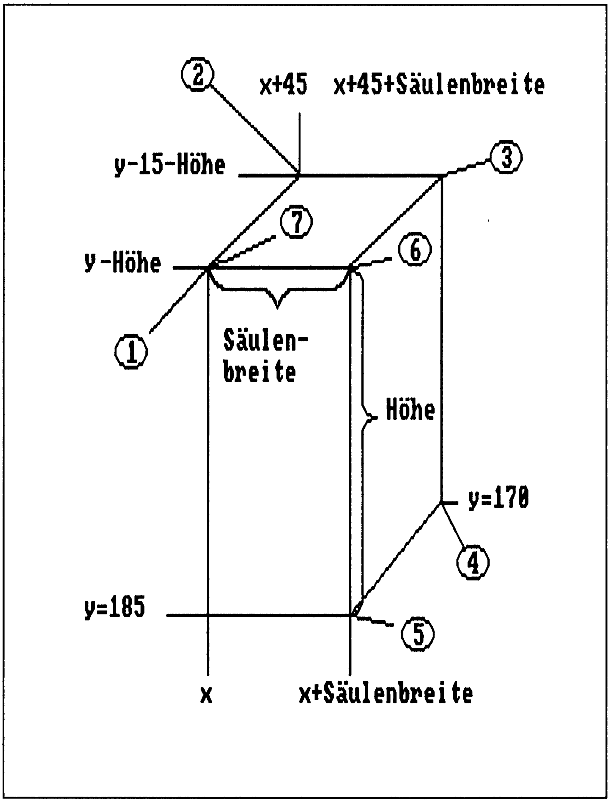


Bild 4.8: Definition der Säulenpunkte

Nachdem mit der AreaEnd-Funktion der Säulenaufbau beendet wurde, wird in der Schleife »saeulenloop« so lange weitergemacht, bis alle Werte auf den Bildschirm gezeichnet sind.

4.6.7 Menü und Zeichensätze setzen und auswerten

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * Nach Grafikaufbau: Menü, Zeichensätze setzen und auswerten

00000D7E mainpl  lea plotmenu,a1
00000D84         move.l iwindowpointer,a0
00000D8A         CALLINT SetMenuStrip            ;Menü setzen
00000D94 waitl   move.l iwindowpointer,a0
00000D9A         move.l 86(a0),a0                ;Warten auf Message
```

```
00000D9E         CALLEXEC GetMsg
00000DA6         tst.l d0
00000DA8         beq wait1                      ;keine
00000DAC         move.l d0,a0
00000DAE         move.l $14(a0),d2              ;Wert holen:
00000DB2         cmpi.l #$200,d2                ;Close-Gadget
00000DB8         beq endplot
00000DBC         cmpi.l #$08,d2
00000DC2         beq maus                       ;Maustaste
00000DC6         clr.l d7
00000DC8         move.w $18(a0),d7              ;Welcher Menüpunkt?
00000DCC         cmpi.w #$ffff,d7               ;keiner
00000DD0         beq wait1
00000DD4         bsr wert                       ;Menüpunkt auswerten
00000DD8         lea tabplot,a4
00000DDE         bra spring                     ;und anspringen

00000DE2 zsatz   dc.l topaz,ruby,sapphire,opal,diamond,garnet,emerald
00000DFE style   dc.l unter,fett,schräg,breit

00000E0E tabplot dc.l zsatz,style               ;Menüs

00000E16 maske   dc.b 0
                 even

         * Text aus Gadget ausgeben nach Maustastendruck

00000E18 maus    move.l iwindowpointer,a0
00000E1E         clr.l d0
00000E20         move.w $0E(a0),d0              ;x-Koordinate
00000E24         move.l d0,setx+2
00000E2A         move.w $0C(a0),d0              ;y-Koordinate
00000E2E         move.l d0,sety+2
00000E34         bsr plprint                    ;Text ausgeben
00000E38         bra wait1                      ;fertig


         * MENÜ ZEICHENSATZ

         * MENÜPUNKT 1 (TOPAZ)

00000E3C topaz   move.l #topazstart,d0          ;Tabellenstart
00000E42         bra mainzei

00000E46 topazstart      dc.l topaz9,topaz8
```

```
         * MENÜPUNKT 2 (RUBY)
00000E4E ruby    move.l #rubystart,d0
00000E54         bra mainzei

00000E58 rubystart       dc.l ruby8,ruby12,ruby15

         * MENÜPUNKT 3 (SAPPHIRE)

00000E64 sapphire  move.l #sapphirestart,d0
00000E6A         bra mainzei

00000E6E sapphirestart   dc.l sapphire14,sapphire19

         * MENÜPUNKT 4 (OPAL)

00000E76 opal    move.l #opalstart,d0
00000E7C         bra mainzei

00000E80 opalstart       dc.l opal9,opal12

         * MENÜPUNKT 5 (DIAMOND)

00000E88 diamond  move.l #diamondstart,d0
00000E8E          bra mainzei

00000E92 diamondstart    dc.l diamond12,diamond20

         * MENÜPUNKT 6 (GARNET)

00000E9A garnet   move.l #garnetstart,d0
00000EA0          bra mainzei

00000EA4 garnetstart     dc.l garnet9,garnet16

         * MENÜPUNKT 7  (EMERALD)

00000EAC emerald  move.l #emeraldstart,d0
00000EB2          bra mainzei

00000EB6 emeraldstart    dc.l emerald17,emerald20

         * Zeichensatz neu setzen

00000EBE mainzei add.l d7,d0                        ;plus Untermenü * 4
00000EC0         move.l d0,a0
00000EC2         move.l (a0),a0                     ;Adresse holen
00000EC4         move.l (a0),a0                     ;Textfont holen
```

```
00000EC6           move.l iwindowpointer,a1          ;Rastport holen
00000ECC           move.l 50(a1),a1
00000ED0           CALLGRAF SetFont                  ;Zeichensatz setzen
00000EDA           bsr clrstyle                      ;Alle Styles löschen
00000EDE           bra wait1                         ;fertig


         * MENÜ STYLE

         * MENÜPUNKT 1 (UNTERSTRICHEN)

00000EE2 unter     bsr holmaske                      ;Maske holen
00000EE6           bchg.b #0,d2                      ;Bit 1 umdrehen
00000EEA           bra setzmaske

         * MENÜPUNKT 2  (FETTDRUCK)

00000EEE fett      bsr holmaske                      ;Maske holen
00000EF2           bchg.b #1,d2                      ;Bit 2 umdrehen
00000EF6           bra setzmaske

         * MENÜPUNKT 3  (SCHRÄGSCHRIFT)

00000EFA schräg    bsr holmaske                      ;Maske holen
00000EFE           bchg.b #2,d2                      ;Bit 3 umdrehen
00000F02           bra setzmaske

         * MENÜPUNKT 4  (BREITSCHRIFT)

00000F06 breit     bsr holmaske                      ;Maske holen
00000F0A           bchg.b #3,d2                      ;Bit 4 umdrehen
00000F0E           bra setzmaske

         * Alte Zeichensatz-Stil-Maske holen (codiert)

00000F12 holmaske  move.l iwindowpointer,a1        ;Rastport holen
00000F18           move.l 50(a1),a1
00000F1C           CALLGRAF AskSoftStyle           ;Maske holen
00000F26           move.b maske,d2
00000F2C           rts

         * Neue Zeichensatz-Stil-Maske setzen

00000F2E setzmaske bsr setmaske                    ;ausführen
00000F32           bra wait1                       ;fertig
```

```
00000F36 setmaske  move.b d2,maske            ;neue Maske retten
00000F3C           move.b d0,d1
00000F3E           move.b d2,d0               ;alles löschen
00000F40           move.l iwindowpointer,a1   ;Rastport holen
00000F46           move.l 50(a1),a1
00000F4A           CALLGRAF SetSoftStyle      ;Maske setzen
00000F54           rts

         * Alle gesetzten Styles löschen

00000F56 clrstyle moveq.l #4,d0               ;4 Menüpunkte
00000F58          lea menuptab,a0             ;Startadresse Struktur
00000F5E clrloop  move.l (a0)+,a1             ;Menüpunkt holen
00000F60          bclr #8,12(a1)              ;CHECKED-Bit löschen
00000F66          subq.w #1,d0
00000F68          bne clrloop
00000F6C          bsr holmaske                ;alle Styles
00000F70          clr.b d2                    ;löschen
00000F72          bra setmaske

         * Ende des Plottems

00000F76 endplot bsr clrstyle                 ;alle Styles löschen
00000F7A         move.l iwindowpointer,a0
00000F80         CALLINT ClearMenuStrip       ;Menü löschen
00000F8A         move.l iwindowpointer,a0
00000F90         CALLINT CloseWindow          ;Window schließen
00000F9A         move.l plotpointer,a0
00000FA0         CALLINT CloseScreen          ;Screen schließen
00000FAA         bra wait

00000FAE menuptab dc.l eintrag1100,eintrag1101,eintrag1102,eintrag1103
```

Nachdem wir unsere Grafik auf den Bildschirm gezaubert haben, möchten wir sie natürlich auch beschriften. Dazu wird ab dem Label »mainpl« zunächst das Zeichensatz- und Style-Menü aktiviert und auf eine Message gewartet, hier wieder mit Hilfe des Pollings. Da wir drei mögliche Ereignisse vorliegen haben (Maustastendruck, Close-Gadget, Menüpunkt gewählt), auf die wir reagieren müssen, wird zunächst das IDCMP-Flag ausgewertet. Das Verfahren zur Menüauswertung dürfte inzwischen bekannt sein, so daß wir uns gleich auf die Maustaste stürzen können.

Der Sinn des Maustastendrucks besteht darin, die Koordinaten festzulegen, ab denen der in dem Stringgadget eingegebene Text ausgegeben werden soll. Man kann sie aus der

Window-Struktur auslesen, die y-Koordiante ab der Adresse $=C und die x-Koordinate ab der Adresse $0E. Das Auslesen und die Textausgabe werden in der Routine »maus« durchgeführt.

Sehr einfach stellt sich die Funktion des Zeichensatz-Menüs dar: Die einzelnen Menüpunkt-Routinen übergeben lediglich den Pointer auf die Adresse, wo der Zeiger auf die jeweilige TextFont-Struktur gespeichert ist, an die Routine mit dem Namen »mainzei«. Hier wird zunächst die Adresse, an der der TextFont-Zeiger gespeichert ist und dann dieser selbst geholt, um die SetTextFont-Routine anspringen zu können und den neuen Zeichensatz zu setzen. Bei der Aktivierung des neuen Zeichensatzes werden automatisch alle Styles gelöscht, da man nicht davon ausgehen kann, daß diese weiterhin gewünscht werden. Diese Tatsache müssen wir aber auch in unserem Style-Menü berücksichtigen, indem alle eventuellen Haken vor den Menüpunkten gelöscht werden. Hierfür ist das Unterprogramm »clrstyle« zuständig, welches die CHECKED-Bits in den vier Menüpunkt-Strukturen löscht.

Das Style-Menü funktioniert ebenso einfach: Da alle Menüpunkte automatisch wechselweise abgehakt werden, müssen wir dies nur noch auf die Style-Maske übertragen. Dazu wird zunächst ab »holmaske« mittels der AskSoftStyle-Funktion die aktuelle Maske geholt. Je nach angewähltem Menüpunkt wird dann das zugehörige Bit umgedreht und die neue Maske der Routine »setmaske« übergeben, welche sie durch die SetSoftStyle-Funktion aktiviert.

Nachdem das Close-Gadget angeklickt wurde, werden zunächst wieder die CHECKED-Bits aus dem Style-Menü gelöscht, da sie bei einem neuen Menüaufbau dazu führen würden, daß einzelne Menüpunkte von Beginn an abgehakt würden. Danach wird das Menü gelöscht und Window sowie Screen geschlossen, womit unser Grafikteil auch schon beendet ist.

4.6.8 Zeichensätze holen

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * Zeichensätze von Disk ins System holen
00000FBE opdifo  lea zeichensatz,a2           ;Start der Struktur
00000FC4         lea ruby8,a3                 ;Start Pointer
00000FCA         move.w #13,d2                ;13 Zeichensätze
00000FCE zloop   move.l a2,a0
00000FD0         CALLDISKFONT OpenDiskFont    ;Zeichensatz holen
00000FDA         move.l d0,(a3)+              ;Zeiger sichern
00000FDC         adda.l #8,a2
00000FE2         subq.w #1,d2
00000FE4         bne zloop                    ;alle Sätze?
00000FE8         rts
```

```
        * Zeichensätze vom ROM ins System holen

00000FEA opfo    lea zeichensatz+112,a0  ;s.o.
00000FF0         lea topaz9,a3
00000FF6         CALLGRAF OpenFont
00001000         move.l d0,(a3)+
00001002         lea zeichensatz+104,a0
00001008         CALLGRAF OpenFont
00001012         move.l d0,(a3)
00001014         rts

        * Zeichensätze verfügbar machen

00001016 adfo    move.w #15,d2                  ;15 Zeichensätze
0000101A         lea ruby8,a3                   ;Start Zeiger
00001020 adloop  move.l (a3),a1
00001022         CALLGRAF AddFont               ;ausführen
0000102C         move.l (a3)+,a1                ;Zeiger erhöhen
0000102E         subq.w #1,d2
00001030         bne adloop
00001034         rts

        * Zeichensätze löschen

00001036 clfo    move.w #13,d2                  ;15 Zeichensätze
0000103A         lea ruby8,a3                   ;Start Zeiger
00001040 clloop  move.l (a3),a1
00001042         CALLGRAF CloseFont             ;ausführen
0000104C         move.l (a3)+,a1                ;Zeiger erhöhen
0000104E         subq.w #1,d2
00001050         bne clloop
00001054         rts


        * Window öffnen und Message auswerten

00001056 mainwindow      CALLINT OpenWindow
00001060                 move.l d0,iwindowpointer

00001066 iloop           move.l iwindowpointer,a0
0000106C                 move.l 86(a0),a0
00001070                 CALLEXEC GetMsg
00001078                 tst.l d0               ;keine Message
0000107A                 beq iloop
0000107E                 move.l d0,a2
00001080                 move.l $14(a2),d2      ;Wert holen:
00001084                 cmpi.l #$200,d2        ;Close-Gadget
```

```
0000108A                  beq close
0000108E                  move.l d0,a2
00001090                  move.l $1C(a2),a2      ;Adresse User-Gadget
00001094                  move.w $26(a2),d2      ;Gadget-ID aus Gadget
00001098                  cmpi.w #111,d2         ;Struktur holen
0000109C                  beq close
000010A0                  cmpi.w #112,d2
000010A4                  bne iloop
000010A8 close            rts
```

Um Fremdzeichensätze benutzen zu können, muß man sie erst in den Speicher holen. Hierfür sind die Unterprogramme »opdifo« für Diskzeichensätze und »opfo« für ROM-Zeichensätze zuständig. Das Funktionsprinzip ist bei beiden Routinen identisch: In einer Schleife wird jeweils der Zeiger auf die TextAttr-Struktur des zu ladenden Zeichensatzes übergeben. Diese Strukturen sind ab dem Label »zeichensatz« (siehe unten) abgelegt. Die Ergebnisse der Funktionen OpenFont bzw. OpenDiskFont, die Zeiger auf die TextFont-Struktur, werden ab »ruby8« (das ist der Name des ersten Zeichensatzes in der Liste) abgelegt.

Um die Zeichensätze nach dem Laden verfügbar zu machen, wird die Routine »adfo« angesprungen: Hier wird ebenfalls in einer Schleife der gerade abgelegte Zeiger auf die TextFont-Struktur genommen, um die AddFont-Funktion anzuspringen. Erst nach der Abarbeitung dieses Unterprogramms kann man einen neuen Zeichensatz benutzen.

Vor dem Programmende muß man unbedingt dafür sorgen, daß der für die Zeichensätze reservierte Speicherplatz wieder freigegeben wird, da man sonst auf Dauer an einer Guru-Medidation nicht vorbeikommt. Besonders schnell kann man sie erhalten, wenn man mehrere Male hintereinander Zeichensätze einlädt und sie dem System mit AddFont hinzufügt. Um solche Effekte zu vermeiden, ruft unser Programm die Routine »clfo« auf, in der alle Fremdzeichensätze aus dem Speicher entfernt werden. Hierfür wird analog zur »adfo«-Routine mit einer Schleife gearbeitet, welche die Zeiger auf die TextFont-Strukturen bereitstellt. Das Unterprogramm »mainwindow« ist völlig identisch zu dem entsprechenden aus dem Wertetabelle-Programm, so daß dazu nichts mehr zu sagen ist.

4.6.9 Umwandlung String - Fließkomma (siehe Wertetabelle)

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * Umwandlung String - Fließkomma

000010AA stringfloat      clr.l float
000010B0                  clr.l d0
000010B2                  clr.l d3
000010B4                  move.l (a1),d5         ;Start Stringstruktur
000010B6                  addi.l #16,d5
```

```
000010BC                move.l d5,a5
000010BE                clr.l d5
000010C0                move.w (a5),d5          ;Länge Text holen
000010C2                move.b (a2),d0          ;erstes Zeichen
000010C4                clr.l d0                ;Position
000010C6                move.l a2,a0
000010C8 strloop2       move.b (a0)+,d3         ;Zeichen holen
000010CA                cmpi.b #".",d3
000010CE                beq found               ;Punkt gefunden
000010D2                addq #1,d0
000010D4                cmp.w d0,d5             ;alles durchsucht
000010D6                beq nopoint
000010DA                bra strloop2

000010DE found          subq #1,d0
000010E0                move.l d0,d4            ;höchster Exponent
000010E2                addq #1,d0
000010E4                move.l a2,a0            ;Punkt aus String
000010E6 strloop3       move.l a0,d1
000010E8                add.l d0,d1             ;eleminieren
000010EA                move.l d1,a6
000010EC                move.b 1(a6),(a6)
000010F0                addq #1,d0
000010F2                cmp.w d0,d5
000010F4                bne strloop3
000010F8                subq #1,d5
000010FA                bra weiter1

000010FE nopoint        move.l d5,d4
00001100                subq #1,d4              ;höchster Exponent

00001102 weiter1        clr.l d6                ;Position
00001104                move.l a2,a5

00001106 strloop4       move.l d4,d0            ;Exponenten holen
00001108                CALLFFP SPFlt           ;in Fließkomma
00001112                move.l d0,d1
00001114                move.l stellentab,d0    ;10 in Fließkomma
0000111A                CALLMATHTRANS SPPow     ;10 hoch x rechnen
00001124                move.l a5,d1            ;Faktor holen (ASCII)
00001126                add.l d6,d1
00001128                move.l d1,a6
0000112A                clr.l d1
0000112C                move.b (a6),d1
0000112E                subi.b #"0",d1          ;minus 48 ("0") =
00001132                move.l d0,d7            ;Zahl in Fließkomma
00001134                move.l d1,d0            ;mit 10 hoch x
```

```
00001136                     CALLFFP SPFlt
00001140                     move.l d7,d1
00001142                     CALLFFP SPMul          ;multiplizieren
0000114C                     move.l float,d1
00001152                     CALLFFP SPAdd          ;zu bisheriger Summe
0000115C                     move.l d0,float        ;addieren
00001162                     subq.l #1,d4           ;Exponenten verringern
00001164                     addq #1,d6             ;Position erhöhen
00001166                     cmp.l d6,d5            ;fertig?
00001168                     bne strloop4
0000116C                     move.l float,d0        ;Negativ-Flag einblenden
00001172                     move.l d0,(a3)
00001174                     move.l (a1)+,d0        ;und Wert in Tabelle
00001176                     move.l a2,d0
00001178                     addi.l #16,d0          ;schreiben
0000117E                     move.l d0,a2
00001180                     move.l (a3)+,d0
00001182                     rts

00001184 float               dc.l 0

00001188 stellentab          dc.b $A0,00,00,$44     ;10 in Fließkomma

         * Tabelle mit Stringanfängen

0000118C strinfotab          dc.l strinfo1,strinfo2,strinfo3,strinfo4,strinfo5
000011A0                     dc.l strinfo6,strinfo7,strinfo8,strinfo9,strinfo10

         * Wandelt Eingaben in Fließkomma

000011B4 wandel              lea strinfotab,a1      ;Tabelle mit Anfängen
000011BA                     move.b #10,d2          ;Anzahl Gadgets
000011BE                     lea puffer,a2
000011C4 strloop5            bsr stringfloat
000011C8                     subq #1,d2
000011CA                     bne strloop5
000011CE                     rts
```

4.6.10 Programmende und Strukturen

HiSoft GenAmiga Assembler 1.21 "Chart-Darstellung"

```
         * ENDE MENÜPUNKTE

000011D0 exit     bra wait11                        ;von vorne
```

```
        *ENDE  PROGRAMM

000011D4 ende    move.l mempoint,a0            ;Speicherplatz
000011DA         move.l #640,d0                ;für FLOOD wieder
000011E0         move.l #256,d1                ;freigeben
000011E6         CALLGRAF FreeRaster

000011F0         bsr clfo                      ;Fonts entfernen
000011F4         move.l windowpointer,a0       ;Menü löschen
000011FA         CALLINT ClearMenuStrip

00001204         move.l windowpointer,a0       ;Window schließen
0000120A         CALLINT CloseWindow

00001214         move.l screenpointer,a0       ;Screen schließen
0000121A         CALLINT CloseScreen

00001224 fini7   move.l _DiskfontBase,a1
0000122A         CALLEXEC CloseLibrary
00001232 fini6   move.l _DOSBase,a1            ;alle Libraries schließen
00001238         CALLEXEC CloseLibrary
00001240 fini5   move.l _GfxBase,a1
00001246         CALLEXEC CloseLibrary
0000124E fini4   move.l _MathTransBase,a1
00001254         CALLEXEC CloseLibrary
0000125C fini3   move.l _IntuitionBase,a1
00001262         CALLEXEC CloseLibrary
0000126A fini2   move.l _MathBase,a1
00001270         CALLEXEC CloseLibrary
00001278 fini1   rts

                 cnop 0,2

0000127A _DOSBase  dc.l 0                      ;Basis-Pointer-Libraries
0000127E _MathBase dc.l 0
00001282 _IntuitionBase  dc.l 0
00001286 _MathTransBase  dc.l 0
0000128A _GfxBase dc.l 0
0000128E _DiskfontBase   dc.l 0

00001292 dosname  dc.b "dos.library",0         ;Namen-Libraries
                 cnop 0,2
0000129E grafname  dc.b "graphics.library",0
000012AF         cnop 0,2
000012B0 ffpname  dc.b "mathffp.library",0
                 cnop 0,2
000012C0 intname  dc.b "intuition.library",0
```

```
                 cnop 0,2
000012D2 mathtransname dc.b "mathtrans.library",0
                 cnop 0,2
000012E4 diskfontname    dc.b "diskfont.library",0
000012F5         cnop 0,2

         * Strukturen
         * a) SCREENS

000012F6 screen_defs     SCREEN 320,200,2,screentitel
00001316 plot_defs       SCREEN 640,400,$8000,plottitel

00001336 screentitel  dc.b "Chart-Darstellung",0
                 cnop 0,4

00001348 plottitel   dc.b "Charts plotten",0
00001357         cnop 0,4

00001358 screenpointer  dc.l 0
0000135C plotpointer     dc.l 0

         * b) WINDOWS

00001360 einwindow WINDOW 0,57,320,143,$60,igadget1,ewindowname,point1,$1006
00001390 diskwindow WINDOW 0,100,320,80,$60,igadget20,dwindowname,point2,
         $10 06
000013C0 startwindow WINDOW20,20,280,180,$360,igadget30,swindowname,point3,
         $1006
000013F0 plotwindow WINDOW 0,0,640,256,$308,igadget40,pwindowname,point4,
         $100E

00001420 ewindowname  dc.b   "Dateneingabe",0
0000142D         cnop 0,2
0000142E dwindowname  dc.b   "Daten laden/speichern",0
                 cnop 0,2
00001444 swindowname  dc.b   "Chart-Darstellung",0
                 cnop 0,2
00001456 pwindowname  dc.b   "Grafische Chartdarstellung",0
00001471         cnop 0,4

00001474 iwindowpointer  dc.l 0
00001478 windowpointer dc.l 0
                         even

         * c) TEXTE

0000147C error        TEXTOUT1 30,7,errortop
```

```
00001490 error1       TEXTOUT1 8,4,errorleft
000014A4 error2       TEXTOUT1 8,4,errorright

000014B8 errortop     dc.b "Ein- Ausgabefehler",0
                      even
000014CC errorleft    dc.b "Nochmal",0
                      even
000014D4 errorright   dc.b "Abbruch",0
                      even

         *Requester aufrufen

000014DC requester    move.l iwindowpointer,a0
000014E2              lea error,a1
000014E8              lea error1,a2
000014EE              lea error2,a3
000014F4              clr.l d0
000014F6              clr.l d1
000014F8              move.l #220,d2
000014FE              move.l #55,d3
00001504              CALLINT AutoRequest
0000150E              rts

00001510 print        PRINTOUT text1,54,20
00001536              PRINTOUT text2,26,40
0000155C              PRINTOUT text3,72,60

00001582              rts


00001584 text1        TEXTOUT text11
00001598 text2        TEXTOUT text22
000015AC text3        TEXTOUT text33

                      even
000015C0 text11       dc.b "Chart-Darstellung V1.0",0
                      even
000015D8 text22       dc.b "(w) 1988 by F.Riemenschneider",0
                      even
000015F6 text33       dc.b "Bitte wählen Sie:",0
                      even

         * Ausgabe des Wahltextes ins Plotfenster

00001608 plprint      lea textplot,a1
0000160E setx         move.l #100,d0
00001614 sety         move.l #100,d1
```

```
0000161A                move.l iwindowpointer,a0
00001620                move.l 50(a0),a0
00001624                CALLINT PrintIText
0000162E                rts

00001630 textplot       TEXTOUT  puffer+16*11


         * d) MENÜS

00001644 menu           MENUE  menu1,10,0,titel0,eintrag000
00001662 menu1          MENUE  0,80,0,titel1,eintrag100

00001680 eintrag000     MENUPUNKT 0,0,0,80,$56,text000,"E",0

000016A4 eintrag100     MENUPUNKT eintrag110,0,0,100,$56,text100,"I",0
000016C8 eintrag110     MENUPUNKT eintrag120,0,15,100,$56,text110,"L",0
000016EC eintrag120     MENUPUNKT 0,0,30,100,$56,text120,"S",0

                        even
00001710 titel0         dc.b "Option",0
                        even
00001718 titel1         dc.b "Daten",0
                        even

0000171E text000        TEXTOUT1 0,0,textt000
00001732 text100        TEXTOUT1 0,0,textt100
00001746 text110        TEXTOUT1 0,0,textt110
0000175A text120        TEXTOUT1 0,0,textt120

                        even
0000176E textt000       dc.b "Ende",0
                        even
00001774 textt100       dc.b "Eingabe",0
                        even
0000177C textt110       dc.b "Laden",0
                        even
00001782 textt120       dc.b "Speichern",0
                        even


0000178C plotmenu       MENUE  plotmenu1,10,0,titel10,eintrag1000
000017AA plotmenu1      MENUE  0,100,0,titel11,eintrag1100

000017C8 eintrag1000    MENUPUNKT eintrag1001,0,0,100,$52,text1000,0,submenu1
000017EC eintrag1001    MENUPUNKT eintrag1002,0,15,100,$52,text1001,0,submenu2
00001810 eintrag1002    MENUPUNKT eintrag1003,0,30,100,$52,text1002,0,submenu3
```

```
00001834 eintrag1003  MENUPUNKT eintrag1004,0,45,100,$52,text1003,0,submenu4
00001858 eintrag1004  MENUPUNKT eintrag1005,0,60,100,$52,text1004,0,submenu5
0000187C eintrag1005  MENUPUNKT eintrag1006,0,75,100,$52,text1005,0,submenu6
000018A0 eintrag1006  MENUPUNKT 0,0,90,100,$52,text1006,0,submenu7

000018C4 submenu1     MENUPUNKT submenu11,100,0,20,$52,text1010,0,0
000018E8 submenu11    MENUPUNKT 0,100,15,20,$52,text1011,0,0
0000190C submenu2     MENUPUNKT submenu21,100,0,20,$52,text1020,0,0
00001930 submenu21    MENUPUNKT submenu22,100,15,20,$52,text1021,0,0
00001954 submenu22    MENUPUNKT 0,100,30,20,$52,text1022,0,0
00001978 submenu3     MENUPUNKT submenu31,100,0,20,$52,text1030,0,0
0000199C submenu31    MENUPUNKT 0,100,15,20,$52,text1031,0,0
000019C0 submenu4     MENUPUNKT submenu41,100,0,20,$52,text1040,0,0
000019E4 submenu41    MENUPUNKT 0,100,15,20,$52,text1041,0,0
00001A08 submenu5     MENUPUNKT submenu51,100,0,20,$52,text1050,0,0
00001A2C submenu51    MENUPUNKT 0,100,15,20,$52,text1051,0,0
00001A50 submenu6     MENUPUNKT submenu61,100,0,20,$52,text1060,0,0
00001A74 submenu61    MENUPUNKT 0,100,15,20,$52,text1061,0,0
00001A98 submenu7     MENUPUNKT submenu71,100,0,20,$52,text1070,0,0
00001ABC submenu71    MENUPUNKT 0,100,15,20,$52,text1071,0,0


00001AE0 eintrag1100  MENUPUNKT eintrag1101,5,0,180,$5F,text1100,"U",0
00001B04 eintrag1101  MENUPUNKT eintrag1102,5,15,180,$5F,text1101,"F",0
00001B28 eintrag1102  MENUPUNKT eintrag1103,5,30,180,$5F,text1102,"S",0
00001B4C eintrag1103  MENUPUNKT 0,5,45,180,$5F,text1103,"B",0


                      even
00001B70 titel10      dc.b "Z-Satz",0
                      even
00001B78 titel11      dc.b "Style",0
                      even

00001B7E text1000     TEXTOUT1 0,0,textt1000
00001B92 text1001     TEXTOUT1 0,0,textt1001
00001BA6 text1002     TEXTOUT1 0,0,textt1002
00001BBA text1003     TEXTOUT1 0,0,textt1003
00001BCE text1004     TEXTOUT1 0,0,textt1004
00001BE2 text1005     TEXTOUT1 0,0,textt1005
00001BF6 text1006     TEXTOUT1 0,0,textt1006
00001C0A text1100     TEXTOUT1 25,0,textt1100
00001C1E text1101     TEXTOUT1 25,0,textt1101
00001C32 text1102     TEXTOUT1 25,0,textt1102
00001C46 text1103     TEXTOUT1 25,0,textt1103
```

```
00001C5A text1010      TEXTOUT1 0,0,textt1010
00001C6E text1011      TEXTOUT1 0,0,textt1011
00001C82 text1020      TEXTOUT1 0,0,textt1020
00001C96 text1021      TEXTOUT1 0,0,textt1021
00001CAA text1022      TEXTOUT1 0,0,textt1022
00001CBE text1030      TEXTOUT1 0,0,textt1030
00001CD2 text1031      TEXTOUT1 0,0,textt1031
00001CE6 text1040      TEXTOUT1 0,0,textt1040
00001CFA text1041      TEXTOUT1 0,0,textt1041
00001D0E text1050      TEXTOUT1 0,0,textt1050
00001D22 text1051      TEXTOUT1 0,0,textt1051
00001D36 text1060      TEXTOUT1 0,0,textt1060
00001D4A text1061      TEXTOUT1 0,0,textt1061
00001D5E text1070      TEXTOUT1 0,0,textt1070
00001D72 text1071      TEXTOUT1 0,0,textt1071

                       even
00001D86 textt1000     dc.b "Topaz",0
                       even
00001D8C textt1001     dc.b "Ruby",0
                       even
00001D92 textt1002     dc.b "Sapphire",0
                       even
00001D9C textt1003     dc.b "Opal",0
                       even
00001DA2 textt1004     dc.b "Diamond",0
                       even
00001DAA textt1005     dc.b "Garnet",0
                       even
00001DB2 textt1006     dc.b "Esmerald",0
                       even

00001DBC textt1100     dc.b "Unterstrichen",0
                       even
00001DCA textt1101     dc.b "Fettschrift",0
                       even
00001DD6 textt1102     dc.b "Schrägschrift",0
                       even
00001DE4 textt1103     dc.b "Breitschrift",0
                       even
                       even
00001DF2 textt1010     dc.b " 8",0
                       even
00001DF6 textt1011     dc.b " 9",0
                       even
00001DFA textt1020     dc.b " 8",0
                       even
```

```
00001DFE texttl021     dc.b "12",0
                       even
00001E02 texttl022     dc.b "15",0
                       even
00001E06 texttl030     dc.b "14",0
                       even
00001E0A texttl031     dc.b "19",0
                       even
00001E0E texttl040     dc.b " 9",0
                       even
00001E12 texttl041     dc.b "12",0
                       even
00001E16 texttl050     dc.b "12",0
                       even
00001E1A texttl051     dc.b "20",0
                       even
00001E1E texttl060     dc.b " 9",0
                       even
00001E22 texttl061     dc.b "16",0
                       even
00001E26 texttl070     dc.b "17",0
                       even
00001E2A texttl071     dc.b "20",0
                       even


         * e) GADGETS

         border
00001E2E          dc.w 0,0        ;x,y-Offset
00001E32          dc.b 3,1        ;Farben
00001E34          dc.b   0        ;Farbmodus
00001E35          dc.b   5        ;Zahl Koordinatenpaare
00001E36          dc.l  koordl    ;Koordinatenpointer
00001E3A          dc.l   0        ;nächster Border

         koordl
00001E3E         dc.w     -2,-2
00001E42         dc.w     122,-2
00001E46         dc.w     122,9
00001E4A         dc.w     -2,9
00001E4E         dc.w     -2,-2

00001E52 string22         dc.b  "Weitere Eingabe",0
                 even
00001E62 string33         dc.b  "Eingabe beenden",0
                 even
```

```
00001E72 string11        dc.b  "---------------------------------------",0
                even
00001E9A string44a       dc.b  "Tortengrafik",0
                even
00001EA8 string44b       dc.b  "Säulengrafik",0
                even
00001EB6 string44c       dc.b  "Startsatz:",0
                even
00001EC2 string44d       dc.b  "Zielsatz:",0
                even
00001ECC string66        dc.b  "Ausführen",0
                even
00001ED6 string77        dc.b  "Abbrechen",0
                even
00001EE0 string88        dc.b  "Filename:",0
                even
00001EEA string99        dc.b  "Eingabetext:",0
                even
00001EF8 string101       dc.b "00",0
                even
00001EFC string111       dc.b "01",0
                even
00001F00 string121       dc.b "02",0
                even
00001F04 string131       dc.b "03",0
                even
00001F08 string141       dc.b "04",0
                even
00001F0C string151       dc.b "05",0
                even
00001F10 string161       dc.b "06",0
                even
00001F14 string171       dc.b "07",0
                even
00001F18 string181       dc.b "08",0
                even
00001F1C string191       dc.b "09",0
                even

00001F20 string1         STRING -27,-11,string11,string2
00001F34 string2         STRING 0,0,string22,0
00001F48 string3         STRING 0,0,string33,0
00001F5C string4a        STRING 12,0,string44a,0
00001F70 string4b        STRING 12,0,string44b,0
00001F84 string4c        STRING 0,-15,string44c,0
00001F98 string4d        STRING 0,-15,string44d,0
00001FAC string6         STRING 23,0,string66,0
```

```
00001FC0 string7          STRING 23,0,string77,0
00001FD4 string8          STRING -90,0,string88,0

00001FE8 string00         STRING -20,0,string101,0
00001FFC string01         STRING -20,0,string111,0
00002010 string02         STRING -20,0,string121,0
00002024 string03         STRING -20,0,string131,0
00002038 string04         STRING -20,0,string141,0
0000204C string05         STRING -20,0,string151,0
00002060 string06         STRING -20,0,string161,0
00002074 string07         STRING -20,0,string171,0
00002088 string08         STRING -20,0,string181,0
0000209C string09         STRING -20,0,string191,0

000020B0 string9          STRING -105,0,string99,0

         * Eingabe-Gadgets

000020C4 igadget1  GADGET igadget2,30,20,1,4,border,string00,strinfo1,1
000020F0 igadget2  GADGET igadget3,30,40,1,4,border,string01,strinfo2,2
0000211C igadget3  GADGET igadget4,30,60,1,4,border,string02,strinfo3,3
00002148 igadget4  GADGET igadget5,30,80,1,4,border,string03,strinfo4,4
00002174 igadget5  GADGET igadget6,30,100,1,4,border,string04,strinfo5,5
000021A0 igadget6  GADGET igadget7,187,20,1,4,border,string05,strinfo6,6
000021CC igadget7  GADGET igadget8,187,40,1,4,border,string06,strinfo7,7
000021F8 igadget8  GADGET igadget9,187,60,1,4,border,string07,strinfo8,8
00002224 igadget9  GADGET igadget10,187,80,1,4,border,string08,strinfo9,7
00002250 igadget10 GADGET igadget11,187,100,1,4,border,string09,strinfo10,9
0000227C igadget11 GADGET igadget12,30,125,1,1,border,string1,0,111
000022A8 igadget12 GADGET 0,187,125,1,1,border,string3,0,112

000022D4 strinfo1          STRINF puffer
000022F8 strinfo2          STRINF puffer+16*1
0000231C strinfo3          STRINF puffer+16*2
00002340 strinfo4          STRINF puffer+16*3
00002364 strinfo5          STRINF puffer+16*4
00002388 strinfo6          STRINF puffer+16*5
000023AC strinfo7          STRINF puffer+16*6
000023D0 strinfo8          STRINF puffer+16*7
000023F4 strinfo9          STRINF puffer+16*8
00002418 strinfo10         STRINF puffer+16*9
0000243C strinfo11         STRINF puffer+16*10
00002460 strinfo12         STRINF puffer+16*11
00002484 strinfo13         STRINF puffer+16*12
000024A8 strinfo14         STRINF puffer+16*13
```

```
         * Startgadgets

000024CC igadget30 GADGET igadget31,70,90,1,1,0,string4a,0,111
000024F8 igadget31 GADGET igadget32,70,115,1,1,0,string4b,0,112
00002524 igadget32 GADGET igadget33,12,155,$801,4,border,string4c,
                          strinfo13,1
00002550 igadget33 GADGET 0,148,155,$801,4,border,string4d,strinfo14,1

         * Diskgadgets

0000257C igadget20 GADGET igadget21,145,20,1,4,border,string8,strinfo11,111
000025A8 igadget21 GADGET igadget22,20,48,1,1,border,string6,0,111
000025D4 igadget22 GADGET 0,175,48,1,1,border,string7,0,112

         * Plot-Gadget

00002600 igadget40 GADGET 0,270,240,1,4,border,string9,strinfo12,1

0000262C puffer           ZAHLEN
000026CC                  dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
000026DC                  dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
000026EC                  dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
000026FC                  dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                          cnop 0,4


0000270C wertestr         ZAHLEN
000027AC                  ZAHLEN
0000284C                  ZAHLEN
000028EC                  ZAHLEN
0000298C                  ZAHLEN
00002A2C                  ZAHLEN
00002ACC                  ZAHLEN
00002B6C                  ZAHLEN
00002C0C                  ZAHLEN
00002CAC                  ZAHLEN
00002D4C werte           ds.l  100
         * Zeichensatzpointer

00002EDC ruby8           dc.l 0
00002EE0 ruby12          dc.l 0
00002EE4 ruby15          dc.l 0
00002EE8 sapphire14      dc.l 0
00002EEC sapphire19      dc.l 0
00002EF0 opal9           dc.l 0
00002EF4 opal12          dc.l 0
00002EF8 diamond12       dc.l 0
```

```
00002EFC diamond20        dc.l 0
00002F00 garnet9          dc.l 0
00002F04 garnet16         dc.l 0
00002F08 emerald17        dc.l 0
00002F0C emerald20        dc.l 0
00002F10 topaz9           dc.l 0
00002F14 topaz8           dc.l 0
00002F18 leer             dc.l 0

         * Zeichensatzstrukturen

00002F1C zeichensatz      ZEICHEN ruby.font,8,0,$62
00002F24                  ZEICHEN ruby.font,12,0,$62
00002F2C                  ZEICHEN ruby.font,15,0,$62
00002F34                  ZEICHEN sapphire.font,14,0,$62
00002F3C                  ZEICHEN sapphire.font,19,0,$62
00002F44                  ZEICHEN opal.font,9,0,$62
00002F4C                  ZEICHEN opal.font,12,0,$62
00002F54                  ZEICHEN diamond.font,12,0,$62
00002F5C                  ZEICHEN diamond.font,20,0,$62
00002F64                  ZEICHEN garnet.font,9,0,$62
00002F6C                  ZEICHEN garnet.font,16,0,$62
00002F74                  ZEICHEN emerald.font,17,0,$62
00002F7C                  ZEICHEN emerald.font,20,0,$62
00002F84                  ZEICHEN topaz.font,8,0,$41
00002F8C                  ZEICHEN topaz.font,9,0,$49
00002F94 zeichenleer      ZEICHEN 0,0,0,0

00002F9C ruby.font        dc.b "ruby.font",0
                          even
00002FA6 sapphire.font    dc.b "sapphire.font",0
                          even
00002FB4 opal.font        dc.b "opal.font",0
                          even
00002FBE diamond.font     dc.b "diamond.font",0
                          even
00002FCC garnet.font      dc.b "garnet.font",0
                          even
00002FD8 emerald.font     dc.b "emerald.font",0
                          even
00002FE6 topaz.font       dc.b "topaz.font",0
                          even

00002FF2 tmpras           ds.l 10
0000301A areainf          ds.l 10
```

Der Aufwand an Strukuren ist bei diesem Programm noch viel höher als bei der Wertetabelle, was wir vor allen Dingen den zahlreichen Menüpunkten und den Zeichensätzen zu verdanken haben. Obwohl die letzten Seiten optisch nicht gerade dazu einladen, ausführlich studiert zu werden, sind hier die wichtigsten Dinge verborgen: Trotz ähnlichem Aussehen der Übergabeparameter sind es oft die Kleinigkeiten, die wesentlich sind!

Das Programm ist so angelegt, daß Sie es sehr einfach erweitern oder Ihren Befürfnissen anpassen können. So könnten Sie z.B. noch eine Kurven- oder 2-D-Tortengrafik mit herausgezogenen Tortenstücken implementieren, eine automatische Beschriftung der Säulen und Torten mit den jeweiligen Werten einführen, usw. Natürlich könnten Sie auch noch weitere Menüs anlegen, in denen man z.B. eine Grafik auf Diskette schreiben oder ausdrucken könnte, die einzige Grenze der Erweiterung wird durch den Speicher festgelegt.

Ich würde mich freuen, vielleicht eines Tages eine expandierte Version dieses Beispielprogramms als Listing des Monats im Amiga-Magazin wiederzufinden (wegen des Copyrights sehe ich keine Probleme, da ja sowohl dieses Buch als auch das Amiga-Magazin im Markt&Technik-Verlag erscheint). Als Anregung könnten Sie sich ja einmal bei einem Bekannten, der einen C64 besitzt, das Programm »Grafic-Calc 64« ansehen.

4.7 Direkte Programmierung der Grafik-Bitplanes

Wie wir in den vorausgegangenen Abschnitten gesehen haben, stellt uns die Graphics-Library eine Reihe an Funktionen zur Verfügung, die normalerweise ausreichen sollte, eine vernünftige Grafik zu programmieren.

Ich hatte Ihnen aber noch eine Kreis-/Ellipsenroutine versprochen, die das Manko der DrawEllipse-Funktion, keine Ausschnitte bearbeiten zu können, behebt. Wie wir im letzten Beispielprogramm gesehen haben, ist es auch nicht weiter schwierig, eine solche Funktion zu programmieren, allerdings blieb die Zeichengeschwindigkeit auf der Strecke.

Der Grund hierfür liegt in den wiederholt aufzurufenden Zeichenfunktionen Draw und WritePixel. Das Problem, daß das Betriebssystem in C und nicht in Assembler programmiert wurde, wird hier deutlich. Bei jedem Aufruf werden nämlich erst einmal gewaltige Datenmengen zwischen Speicher, Registern und Stapel verschoben, bevor die eigentliche Routine aufgerufen wird. Die DrawEllipse-Funktion hingegen verschiebt nur einmal beim Aufruf, der grundlegende Unterschied besteht dort darin, daß zwar auch dort die Punktsetzroutinen angesprungen werden, aber ohne großartige Verschiebemaßnahmen, denn die Parameter für die Koordinaten werden nicht von außen, sondern routinenintern übergeben.

Dieses Problem ist fundamental in der Sprache C, bzw. deren Kompilaten begründet. Man sollte daher, wann immer es geht, wiederholte Aufrufe von Library-Funktionen vermeiden. Zur Not muß man auch einmal eine eigene Routine programmieren. Wie gesagt liegt für uns das Zeitproblem in den Punktsetzroutinen begründet. Wir wollen daher, bevor wir auf den eigentlichen Ellipsen-Algorithmus zu sprechen kommen, zunächst eine eigene Plot-

Routine entwerfen, die wir danach von der Kreis-/Ellipsenroutine aufrufen können. War man auf dem C64 noch durch fehlende Grafikroutinen gezwungen, den Grafikspeicher direkt anzusprechen, so ist auf dem Amiga die Programmiersprache des Betriebssystems der Grund. Ich frage mich, ob es eines Tages Commodore gelingen wird, vielleicht ein Assembler-programmiertes Betriebssystem herauszubringen, an dem man ohne Wenn und Aber seine Freude haben wird.

4.7.1 Aufbau des Grafikspeichers

Der Aufbau einer Bitplane ist an sich trivial: Man hat eine bestimmte Anzahl von Bits, von denen jedes einzelne einen Grafikpunkt repräsentiert. Das Grundproblem besteht nun zum einen darin, einen Algorithmus zu entwerfen, der einen Zusammenhang zwischen einer bestimmten (x,y-)Koordinate und einem Grafikbit herstellt, zum anderen in dem Zusammenhang zwischen einer bestimmten Farbe und den Bitplanes.

Zunächst aber zu den Koordinaten. Da es der Mensch nicht gewohnt ist, in Bits und Bytes zu denken wie unser Computer, sondern mit x- und y-Koordinaten rechnet, muß man einen Weg der Umrechnung finden. Dazu muß man sich erst einmal überlegen, wohin man den Koordinatenursprung legt. In der Schule legt man ihn meistens in die Tafelmitte und rechnet dann mit positiven und negativen Koordinaten. Um die negativen Zahlen zu vermeiden, hat man es sich in der Computerwelt angewöhnt, den Ursprung in die linke obere Ecke des Bildschirms zu legen. Die Graphics-Library geht, wie wir gesehen haben, ebenfalls von dieser Vereinbarung aus. Man muß sich aber immer darüber im klaren sein, daß es sich wirklich nur um eine Vereinbarung handelt, genausogut hätte man jeden anderen Bildschirmpunkt als Ursprung definieren können. Um den Zusammenhang zwischen Bits und Koordinaten herzustellen, wollen wir uns zunächst einmal Bild 4.9 ansehen, in dem der prinzipielle Aufbau einer Bitplane dargestellt ist. Man sieht, daß die Grafikbytes zeilenweise angeordnet sind, nicht wie beim C64 in 8*8 Byte-Blöcken. Der Punktsetz-Algorithmus fällt deshalb auch relativ einfach aus.

Die Anfangsadresse einer Zeile bekommt man einfach, indem man die Anzahl der x-Punkte durch 8 geteilt so oft zu der Startadresse der Bitplane hinzuaddiert, wie es durch die y-Koordinate festgelegt wird. Falls wir also 320 Punkte pro Zeile haben, würde sich die Adresse der zu der y-Koordinate 6 gehörenden Grafikzeile durch die Rechnung

Adresse = Start + (320/8)*6 = Start + 240

bestimmen lassen. Jetzt fehlt uns nur noch das entsprechende Byte innerhalb der Zeile. Da jeweils 8 Punkte in einem Byte liegen, können wir die Adresse des Grafikbytes durch folgende Rechnung bestimmen, z.B. für x = 13:

Adresse = Zeilenstart + INT(13/8) = Zeilenstart + 1

Dies ist nicht schwer einzusehen, da das erste Byte einer Zeile für die Grafikpunkte 0-7, das zweite für die Punkte 8-15 zuständig ist, usw. Auf eine Bitplane bezogen können wir

nun den Offset für den Punkt (x,y) = (13,6) berechnen, indem wir beide Gleichungen zusammenfassen:

Adresse = Start + (320/8)*6 + INT(13/8) = 241

Als letztes ist es nun erforderlich, das für den Punkt zuständige Bit aus dem Grafikbyte herauszufiltern. Dazu werden wir eine Tabelle anlegen, die den Wert des relvanten Bits für alle 320 (640 in HiRes-Modus) möglichen x-Koordinaten enthält, z.B. 128 (=Bit 7) für x=0,8,16,... oder 32 (=Bit 5) für x=2,10,18,...

Mit diesen Informationen ist es kein Problem mehr, einen Grafikpunkt in einer bestimmten Bitplane zu zeichnen. Man muß sich nun aber noch über die Farben Gedanken machen. Wie in Kapitel 2 ausgeführt, ergibt sich eine bestimmte Farbe aus einer Kombination von gesetzten und gelöschten Bits in den verschiedenen Bitplanes. Da die relative Position zur Startadresse der Bitplanes aber immer konstant bleibt, braucht man den Offset für die Position des Grafikbytes nur einmal zu berechnen und muß dann, abhängig von der gewünschten Farbe, die einzelnen Bits löschen oder setzen.

Bei einer Figur wie einer Ellipse hat man jedoch den Vorteil, daß alle Punkte in der gleichen Farbe gesetzt werden. Wenn man die Zeichenfarbe nun noch so geschickt wählt, daß man nur auf eine Bitplane zugreifen muß, erreicht man natürlich eine maximale Zeichengeschwindigkeit. Voraussetzung für das einwandfreie Funktionieren dieses Prinzips ist allerdings, daß die entsprechenden Bits in den anderen Bitplanes gelöscht sind. Dies ist immer dann der Fall, wenn an dieser Stelle vorher noch kein Punkt in einer anderen Farbe als der Zeichenfarbe der Ellipse gesetzt wurde. Falls Sie nur mit einer Bitplane arbeiten, brauchen Sie sich überhaupt keine Gedanken mehr zu machen.

Um diesen Worten die Tat folgen zu lassen, müssen wir als erstes die Startadresse einer Bitplane kennen. Diese ist in der sogenannten Bitplane-Struktur versteckt. Ab der Adresse 8 folgen dort nacheinander die Langwörter mit den Bitplane-Zeigern, wobei die Anzahl natürlich von der Zahl der Bitplanes abhängt.

Den Zeiger auf die Bitplane-Struktur erhalten wir aus der Rast-Port-Struktur des Screens (nicht wie gewohnt des Windows!) ab der Adresse 4. Da sich die Rast-Port-Struktur nach den Aussagen in Kapitel 2 ihrerseits ab der Adresse $54 innerhalb der Screen-Struktur befindet, können wir den Zeiger ab der Adresse $54+4=$58 auslesen. Um z.B. die Adresse der ersten Bitplane zu erhalten, müßten nachfolgende Programmzeilen ausgeführt werden:

```
move.l screenpointer,a0    ;Adresse 1. Bitmap
move.l $58(a0),a0          ;holen
move.l $08(a0),a4
```

Es ist ganz wichtig, daß Sie diesen Pointer in einer Variablen speichern, die in Ihrem Programm sonst nicht benutzt wird. Falls nämlich eine Veränderung stattfindet, kann es passieren, daß Sie unversehens in Speicherbereiche hineinschreiben, die nicht für Grafikpunkte, dafür aber für Guru-Meditationen zuständig sind. Jetzt können wir aber endlich zu unserer Plot-Routine kommen, die neben dem Setzen eines Punktes auch das Löschen und

das Invertieren beherrscht. Dafür muß in dem sogenannten Modus-Flag die Betriebsart festgelegt werden:

Modus	Betriebsart
0	Punkt setzen
128–255	Punkt löschen
1–127	Punkt invertieren

Nun aber zu unserer Plot-Routine, die die x- und y-Koordinate im Wort-Format in den Registern D0 (x) und D1 (y) erwartet, und für den HIRES-Screen ausgelegt ist. Falls Sie einen Screen mit nur 320 Punkten pro Zeile verwenden, müssen Sie den Wert in Zeile »screen« von 80 auf 40 abändern (40 Grafikbyte pro Zeile). Der Pointer auf die Bitplane wird im Register A4 vorausgesetzt:

```
         * Punkt setzen/löschen/invertiern

plot     and.l #$FFFF,d0          ;INT(X/8) = Byte in Zeile
         move.w d0,a1
         divu #8,d0
         and.l #$FF,d1            ;y*80 = Zeilenoffset
screen   mulu #80,d1
         add.l a4,d1              ;Start Bitmap
         add.w d0,d1
         move.l a1,d0
         move.b (a2,d0.w),d0      ;Bit holen
         move.l d1,a1
         move.b (a1),d1           ;Grafikbyte holen
         tst.b modus              ;Modus holen
         beq setz                 ;Punkt setzen
         bmi lösch                ;Punkt löschen

         * Punkt invertieren

         eor.b d0,d1              ;Bit invertieren
         move.b d1,(a1)           ;Byte setzen
         rts

         * Punkt setzen

setz     or.b d0,d1               ;Bit setzen
         move.b d1,(a1)           ;Byte setzen
         rts
```

```
         * Punkt löschen

lösch    eor.b #$FF,d0            ;Byte umdrehen
         and.b d0,d1              ;Bit löschen
         move.b d1,(a1)           ;Byte setzen
         rts


         * Bits der x-Koordinaten

bittab           dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1

                 insgesamt 40 äquivaltente Zeilen

                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
```

Die Routine arbeitet genauso, wie wir es oben in Form der Gleichungen beschrieben haben. Die Tabelle »bittab« habe ich hier nicht ausgeschrieben, da sie aus 40 gleich aufgebauten Zeilen mit jeweils 16 Werten (ingesamt 640) besteht. Zu klären ist nur noch die Funktion der Setz-, Lösch- und Invertierroutine.

Bevor die einzelnen Programmteile angesprungen werden, befinden sich im Register D1 das Grafikbyte, auf das zugegriffen wird, sowie im Register D0 der Wert des für den Grafikpunkt zuständigen Bits. Falls nun ein Punkt gesetzt werden soll, muß dieses Bit zusätzlich zu den anderen schon im Grafikbyte gesetzten Bits eingeblendet werden, was mit der logischen OR-Funktion geschehen kann. Soll der Punkt invertiert werden, muß das schon im Grafikbyte an dieser Position stehende Bit umgedreht werden. Dazu kann man einfach die EXOR-Funktion benutzen: Da alle anderen Bits im Register D0 gelöscht sind, kann innerhalb des Grafikbytes keine Wirkung auf die entsprechenden Bits ausgeübt werden.

Am kompliziertesten ist das Löschen eines Bits, da man dies nur mit der And-Funktion erreichen kann. Um alle anderen Bits im Grafikbyte aber nicht zu beeinflussen, müssen zunächst alle Bits des Verknüpfungsbytes umgedreht werden, was mit dem EOR #$FF-Befehl erreicht wird. Hiernach sind alle Bits, außer dem für den Grafikpunkt zuständigen Bit, das gelöscht ist, gesetzt. Durch die Verknüpfung mit dem Grafikbyte durch die And-Funktion wird nun das für den Punkt verantwortliche Bit gelöscht, während alle anderen Bits unbeeinflußt bleiben. Diese Routine können Sie natürlich auch in Ihre eigenen Programme einbauen, ohne die nun folgende Ellipsenroutine aufrufen zu müssen. Es gibt auch bei der Amiga-Grafik noch Defizite, die Sie beheben können: So wäre eine Textfunktion, mit der man nicht nur waagerecht, sondern auch in anderen Winkeln schreiben kann, sicherlich eine sinnvolle Programmieraufgabe. Ebenso nützlich sind Funktionen zur Manipulation von Grafikausschnitten, wie Verkleinern, Vergrößern, Rotation, etc., die man in einem Mal- oder Konstruktionsprogramm benötigt.

4.7.2 Die Turbo-Kreis/-Ellipsenroutine

Wie schon angekündigt, werden Sie nun eine der schnellsten und universellsten Ellipsenroutinen kennenlernen, die bis heute für einen Heim-Computer entwickelt wurden. Sie setzt ca. 12 600 Punkte pro Sekunde, wobei allerdings einige Punkte doppelt gezeichnet werden, was am Funktionsprinzip dieses Verfahrens liegt. Das Schöne ist vor allen Dingen, daß die Zeichengeschwindigkeit völlig unabhängig von der Größe der Radien ist und nicht, wie man dieses immer gewohnt ist, mit der Größe der Ellipse quadratisch zunimmt (siehe DrawEllipse...)

Spalte	00000000	00111111	11112222	22222233	33333333	....	66666666
	01234567	89012345	67890123	45678901	23456789	....	33333333
							23456789
Bits	76543210	76543210	76543210	76543210	76543210	....	76543210
Zeile 00	Ad	Ad +1	Ad +2	Adr+3	Ad +4	....	Ad +79
Zeile 01	Ad +80	Ad +81	Ad +82	Ad +83	Ad +84	....	Ad +159
Zeile 02	Ad +160	Ad +161	Ad +162	Ad +163	Ad +164	....	Ad +239
Zeile 03	Ad +240	Ad +241	Ad +242	Ad +243	Ad +244	....	Ad +319
Zeile 04	Ad +320	Ad +321	Ad +322	Ad +323	Ad +324	....	Ad +399
Zeile 05	Ad +400	Ad +401	Ad +402	Ad +403	Ad +404	....	Ad +479
..	..	..	..	..	..	..	..
Zeile 251	Ad+20080	Ad+20081	Ad+20082	Ad+20083	Ad+20084	....	Ad+20159
Zeile 252	Ad+20160	Ad+20161	Ad+20162	Ad+20163	Ad+20164	....	Ad+20239
Zeile 253	Ad+20240	Ad+20241	Ad+20242	Ad+20243	Ad+20244	....	Ad+20319
Zeile 254	Ad+20320	Ad+20321	Ad+20322	Ad+20323	Ad+20324	....	Ad+20399
Zeile 255	Ad+20400	Ad+20401	Ad+20402	Ad+20403	Ad+20404	....	Ad+20479

*Bild 4.9 a : Aufbau einer Bitplane im HIRES-Modus (640*256)*
(Ad : Startadresse der Bitplane)

Spalte	00000000	00111111	11112222	22222233	33333333	....	66666666
	01234567	89012345	67890123	45678901	23456789	....	33333333
							23456789
Bits	76543210	76543210	76543210	76543210	76543210	....	76543210
Zeile 00	Ad	Ad +1	Ad +2	Adr+3	Ad +4	....	Ad +79
Zeile 01	Ad +80	Ad +81	Ad +82	Ad +83	Ad +84	....	Ad +159
Zeile 02	Ad +160	Ad +161	Ad +162	Ad +163	Ad +164	....	Ad +239
Zeile 03	Ad +240	Ad +241	Ad +242	Ad +243	Ad +244	....	Ad +319
Zeile 04	Ad +320	Ad +321	Ad +322	Ad +323	Ad +324	....	Ad +399
Zeile 05	Ad +400	Ad +401	Ad +402	Ad +403	Ad +404	....	Ad +479
..	..	..	..	..	..	..	..
Zeile 507	Ad+40560	Ad+40561	Ad+40562	Ad+40563	Ad+40564	....	Ad+40639
Zeile 508	Ad+40640	Ad+40641	Ad+40642	Ad+40643	Ad+40644	....	Ad+40719
Zeile 509	Ad+40720	Ad+40721	Ad+40722	Ad+40723	Ad+40724	....	Ad+40799
Zeile 510	Ad+40800	Ad+40801	Ad+40802	Ad+40803	Ad+40804	....	Ad+40879
Zeile 511	Ad+40880	Ad+40881	Ad+40882	Ad+40883	Ad+40884	....	Ad+40959

*Bild 4.9 b: Aufbau einer Bitplane im HIRES/INTERLACE-Modus (640*512)*
(Ad : Startadresse der Bitplane)

Spalte	00000000	00111111	11112222	22222233	33333333	....	33333333
	01234567	89012345	67890123	45678901	23456789	....	11111111
							23456789
Bits	76543210	76543210	76543210	76543210	76543210	....	76543210
Zeile 00	Ad	Ad +1	Ad +2	Adr+3	Ad +4	....	Ad +39
Zeile 01	Ad +40	Ad +41	Ad +42	Ad +43	Ad +44	....	Ad +79
Zeile 02	Ad +80	Ad +81	Ad +82	Ad +83	Ad +84	....	Ad +119
Zeile 03	Ad +120	Ad +121	Ad +122	Ad +123	Ad +124	....	Ad +159
Zeile 04	Ad +160	Ad +161	Ad +162	Ad +163	Ad +164	....	Ad +199
Zeile 05	Ad +200	Ad +201	Ad +202	Ad +203	Ad +204	....	Ad +239
..	..	..	..	..	..	..	..
Zeile 251	Ad+10040	Ad+10041	Ad+10042	Ad+10043	Ad+10044	....	Ad+10079
Zeile 252	Ad+10080	Ad+10081	Ad+10082	Ad+10083	Ad+10084	....	Ad+10119
Zeile 253	Ad+10120	Ad+10121	Ad+10122	Ad+10123	Ad+10124	....	Ad+10159
Zeile 254	Ad+10160	Ad+10161	Ad+10162	Ad+10163	Ad+10164	....	Ad+10199
Zeile 255	Ad+10200	Ad+10201	Ad+10202	Ad+10203	Ad+10204	....	Ad+10239

*Bild 4.9 c : Aufbau einer Bitplane im Normal-Modus (320*256)*
(Ad : Startadresse der Bitplane)

```
Spalte     |00000000|00111111|11112222|22222233|33333333|........|33333333|
           |01234567|89012345|67890123|45678901|23456789|........|11111111|
           |        |        |        |        |        |        |23456789|
           |        |        |        |        |        |        |        |

Bits       |76543210|76543210|76543210|76543210|76543210|........|76543210|
           |        |        |        |        |        |        |        |

Zeile 00   |Ad      |Ad +1   |Ad +2   |Adr+3   |Ad +4   |........|Ad +39  |
Zeile 01   |Ad +40  |Ad +41  |Ad +42  |Ad +43  |Ad +44  |........|Ad +79  |
Zeile 02   |Ad +80  |Ad +81  |Ad +82  |Ad +83  |Ad +84  |........|Ad +119 |
Zeile 03   |Ad +120 |Ad +121 |Ad +122 |Ad +123 |Ad +124 |........|Ad +159 |
Zeile 04   |Ad +160 |Ad +161 |Ad +162 |Ad +163 |Ad +164 |........|Ad +199 |
Zeile 05   |Ad +200 |Ad +201 |Ad +202 |Ad +203 |Ad +204 |........|Ad +239 |
  ..          ..       ..       ..       ..       ..       ..       ..
Zeile 507  |Ad+20280|Ad+20281|Ad+20282|Ad+20283|Ad+20284|........|Ad+20319|
Zeile 508  |Ad+20320|Ad+20321|Ad+20322|Ad+20323|Ad+20324|........|Ad+20359|
Zeile 509  |Ad+20360|Ad+20361|Ad+20362|Ad+20363|Ad+20364|........|Ad+20399|
Zeile 510  |Ad+20400|Ad+20401|Ad+20402|Ad+20403|Ad+20404|........|Ad+20439|
Zeile 511  |Ad+20440|Ad+20441|Ad+20442|Ad+20443|Ad+20444|........|Ad+20479|
```

*Bild 4.9 d. : Aufbau einer Bitplane im INTERLACE-Modus (320*256)*
(Ad : Startadresse der Bitplane)

Bevor wir uns aber näher mit diesem Verfahren beschäftigen, wollen wir uns einmal den Aufbau einer Ellipse ansehen (Bild 4.10). Um die Koordinaten der einzelnen Punkte zu berechnen, gibt es mehrere Möglichkeiten. Hier haben wir die sogenannten Zylinderkoordinaten gewählt, mit denen meistens auch gerechnet wird. Wenn man den Mittelpunkt der Ellipse durch die Koordinaten XM und YM beschreibt, den Radius in x-Richtung mit XR und denselben in y-Richtung mit YR, und schließlich den Winkel W einführt, kann man eine Ellipse als die Punktmenge definieren, die diese Gleichungen erfüllt:

X = XM + XR*COS(W) 0 Grad < W < 360 Grad
Y = YM + YR*SIN(W) 0 Grad < W < 360 Grad

Falls XR und YR gleich sind, liegt der Sonderfall eines Kreises vor. Man braucht also tatsächlich nur eine Routine, um sowohl Kreise als auch Ellipsen zeichnen zu können.

Das übliche Verfahren, um eine Ellipse zu zeichnen, haben wir bereits in unserem Chart-Grafik-Programm demonstriert: Für jeden Winkel werden die entsprechenden Sinus- und Cosinus-Werte berechnet, mit den Radien multipliziert und in eine Integer-Zahl umgewandelt, die zu den Mittelpunktskoordinaten hinzuaddiert wird. Ein Problem besteht darin, daß bei einer idealen Ellipse zwei nebeneinanderliegende Winkel W unendlich dicht zusammen liegen, d.h., es sind theoretisch unendlich viele Berechnungsschritte erforderlich. Man hilft sich wie gesehen damit, daß nur einige Punkte berechnet und durch Linien verbunden

werden. Je größer diese Schrittweite ist, desto schneller kann die Ellipse aufgebaut werden, desto größer ist aber auch die Abweichung von einer idealen Ellipse.

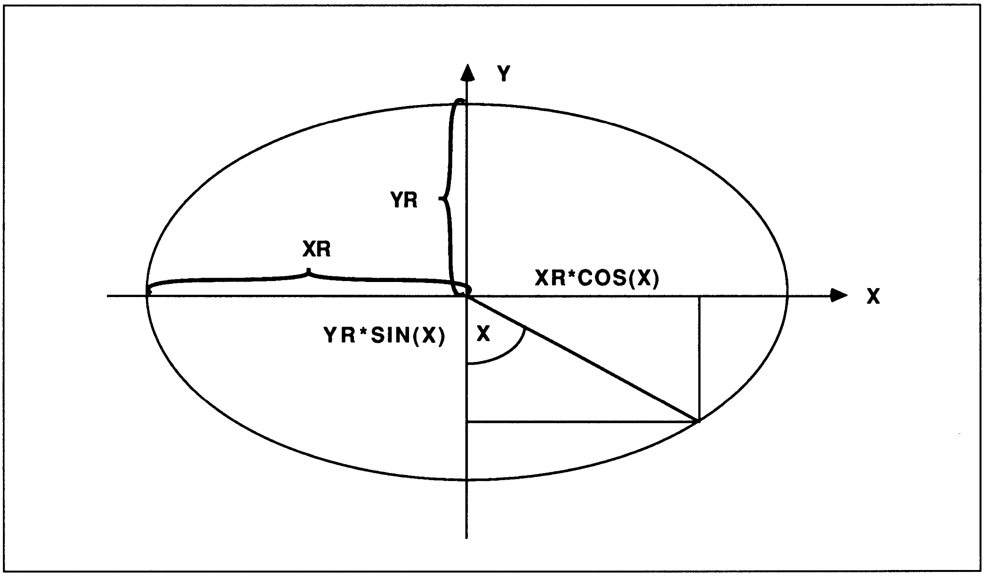


Bild 4.10: Aufbau einer Ellipse

Es ist klar, daß man auf dem Amiga keine solche ideale Ellipse darstellen kann, da die Grafikauflösung nicht unendlich groß ist, sondern maximal nur 640*512 Punkte beträgt. Die genaueste mögliche Ellipse erhält man, wenn zwei nebeneinanderliegende x- und y-Werte jeweils nicht mehr als einen Grafikpunkt auseinanderliegen. Dies ist das Amiga-Kriterium, an dem man den Begriff »Genauigkeit« definitiv festhalten kann.

In der Praxis heißt dies, daß man eine Schrittweite von ca. 0.1 Grad nehmen muß, um dieser Forderung zu entsprechen. Leider wird dadurch die Zeichengeschwindigkeit aber unakzeptabel hoch. Daher überlegte man sich ein Optimierungsverfahren, das auch von der DrawEllipse-Funktion verwendet wird. Hierbei werden nämlich mathematische Beziehungen zwischen den Sinus- und Cosinus-Werten bestimmter Winkel ausgenutzt. Es gilt:

1: SIN(W) = SIN(180 Grad - W) = -SIN(180 Grad + W) = -SIN(360 Grad - W)
2: COS(W) = COS(360 Grad - W) = -COS(180 Grad - W) = -COS(180 Grad + W)

Durch diese Tatsache ist es möglich, mit einem berechneten Wertepaar jeweils in jedem Quadranten einen Punkt zu setzen, da sie sich nur im Vorzeichen unterscheiden. Wenn wir den Term XR*COS(W) mit XOFF und YR*SIN(W) mit YOFF bezeichnen, ergeben sich für die berechneten Koordinaten der einzelnen Quadranten folgende Zusammenhänge:

1. Quadrant (0–90 Grad): X = XM+XOFF; Y = YM-^YOFF
2. Quadrant (90–180 Grad): X = XM+XOFF; Y = YM+YOFF
3. Quadrant (180–270 Grad): X = XM-XOFF; Y = YM+YOFF
4. Quadrant (270–360 Grad): X = XM-XOFF; Y = YM-YOFF

Durch diese Methode konnte die DrawEllipse-Routine die Rechenzeit ca. um den Faktor 4 verkleinern. Allerdings ist sie nun nicht mehr in der Lage, beliebige Ellipsenausschnitte zu zeichnen, wie wir es zur Erstellung des unteren Tortenrandes gebraucht hätten.

Trotz dieses Kunstgriffs ist es aber nicht gelungen, die Rechenzeit soweit zu verkürzen, daß man den Ellipsenaufbau optisch nicht mehr mitverfolgen kann, was für mich das Kriterium für Geschwindigkeit ist. Das eigentliche Problem besteht nach wie vor in der Verwendung von Fließkommazahlen. Eigentlich ist deren Verwendung als paradox zu bezeichnen, da bei der Grafik ja nur Integer-Werte benutzt werden können. Wie aber soll es sonst möglich sein, die Sinus- und Cosinus-Werte zu berechnen? Die entscheidene Antwort ist: überhaupt nicht. Wir werden nun ein 1987 entwickeltes Verfahren benutzen, das in dem C64-Buch »Alles über Maschinensprache« veröffentlicht wurde und schon auf dem Rechenzwerg C-64 für revolutionäre Ergebnisse sorgte. Die Idee dieses Verfahrens besteht darin, die Sinus- und Cosinus-Werte in einer Integer-Tabelle abzulegen. Dies ist so unmittelbar nicht möglich, da sie nur im Bereich von 0 bis 1 liegen. Wenn man sie aber mit einer hinreichend großen Zahl multipliziert, steht der Umwandlung ins Integer-Format nichts mehr im Wege. Um mit dem Typ Byte arbeiten zu können, was die maximale Geschwindigkeit zuläßt, werden wir hierfür den Wert 255 nehmen. Daher gilt für unsere Tabelle:

```
Wert = INT(COS(W)*255)
```

Wenn wir dann den Offset berechnen wollen, müssen wir

```
XOFF = XM + XR*(Wert/255)
```

rechnen. An dieser Stelle müssen wir uns leider wieder mit dem Problem der Genauigkeit auseinandersetzen. Wie viele Werte soll unsere Tabelle enthalten und in welcher Schrittweite müssen diese angelegt sein?

Wir benötigen die Werte von 0 Grad < W < 90 Grad, mit denen wir ja auch die anderen drei Quadranten erschlagen können (s.o.). Unsere Forderung ist die, daß zwei benachbarte Werte um maximal die Zahl 1 differieren dürfen, um die größtmögliche darstellbare Genauigkeit auf dem Amiga zu erhalten. Hier hilft nur ausprobieren: Es zeigt sich, daß die Schrittweite 1/391 die größte mögliche ist, die diese Forderung erfüllt. Eine geringere Schrittweite bringt einen Zuwachs an Rechenzeit, eine größere einen Verlust an Genauigkeit. Wir haben nach dem Grundsatz »so grob wie möglich, so fein wie nötig« die optimale Schrittweite erhalten. Unsere Tabellenwerte können wir nun in einer Schleife berechnen:

```
FOR I = 0 TO 391: WERT(I) = 255*INT(COS(90/391)*I)): NEXT I
```

Wir haben es also geschafft, den Übergang vom Winkel W in eine Schleife von Integer-Werten zu erreichen. Die kritischste Stelle der Genauigkeit liegt übrigens beim Cosinus bei 90 Grad, da er dort seine größte Steigung aufweist, beim Sinus bei 0 Grad.

Wie können wir nun auf einfachste Weise den X-Offset berechnen? Wir müssen einfach in einer Schleife von 0 bis 391 die jeweiligen Werte auslesen, mit dem X-Radius multiplizieren und durch 255 teilen:

```
FOR I = 0 TO 391: X = XM + XR*(Wert(I)/255): NEXT I
```

Hier der Beweis: Wenn wir die Berechnungsgrundlage für die Tabellenwerte einsetzen, ergibt sich folgendes:

```
FOR I = 0 TO 391: X = XM + XR*(255*INT(COS(90/391)*I)/255): NEXT I
```

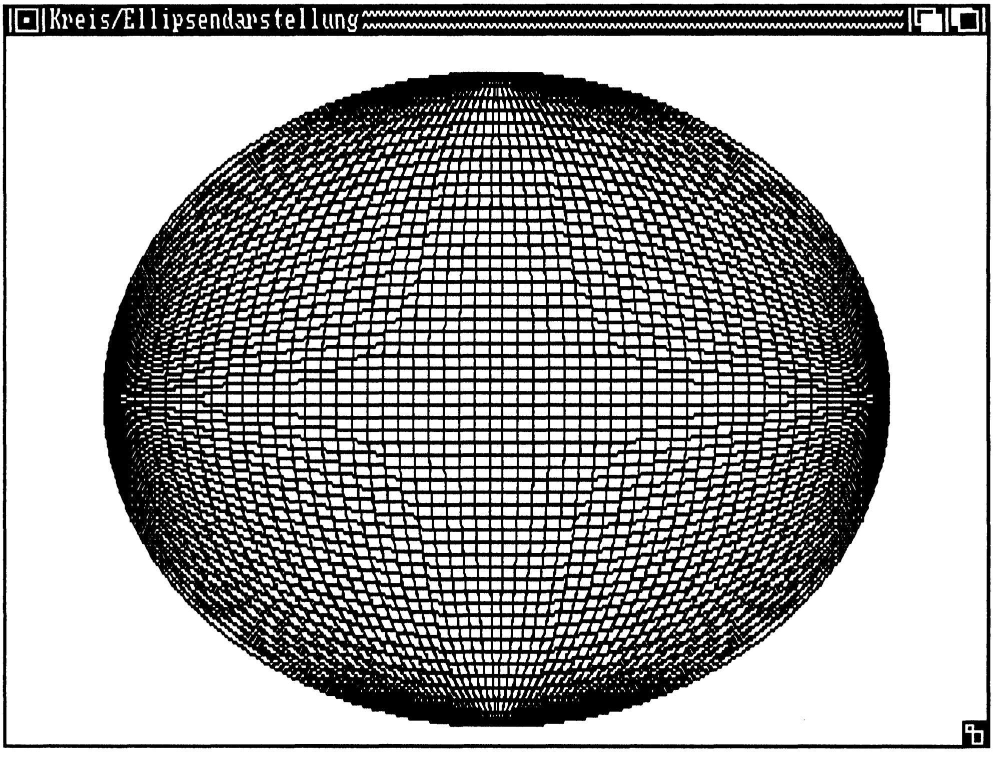


Bild 4.11: 33000 Grafikpunkte in 6,7 Sekunden gezeichnet

Dies kann man aber auch so ausdrücken:

```
FOR I = 0 TO 90 : X = XM + XR*INT(COS(I)): NEXT I
```

Das einzige Problem besteht darin, daß nur Radien bis einschließlich zum Wert 255 zugelassen werden können. Bis zu diesem Zeitpunkt nämlich kann der Ausdruck

```
XOFF = XR*(Wert(I)/255)
```

nicht größer als Eins werden, d.h., unsere gewünschte Genauigkeit ist gegeben. Mit dieser Einschränkung kann man aber leben, wenn man bedenkt, daß der maximale Durchmesser in x-Richtung 640 und in y-Richtung 512 Punkte beträgt, in x- und y-Richtung aber jeweils 2*255 = 510 Punkte zulässig sind. Für die Sinus-Werte brauchen wir keine zusätzliche Tabelle anzulegen, da gilt:

```
SIN(W) = COS(90 Grad - W)
```

Wir können unsere Tabelle daher einmal von vorne für die Cosinus-Werte und einmal von hinten für die Sinus-Werte benutzen. Unsere Routine setzt im Normalfall ebenso vier Punkte pro Wert wie die DrawEllipse-Funktion. Falls aber keine ganze Ellipse gewünscht wird, werden spezielle Routinen angesprungen, die immer nur einen Punkt pro Quadrant setzen. Durch die Kombination dieser Routinen ist es möglich, wirklich beliebige Ausschnitte zu zeichnen. Bevor ich noch einige Worte zu den Übergabeparametern sage, wollen wir uns zunächst den Quelltext des Demoprogramms und die fertige Kreis/Ellipse (Bild 4.11) ansehen:

HiSoft GenAmiga Assembler 1.21 "Kreis/Ellipse"

```
                opt l-,d+,s-,n+
                plen 72
                llen 80

* INCLUDE-Files laden

                include df0:include/exec/exec_lib.i
                include df0:include/math/mathffp_lib.i
                include df0:include/intuition/intuition_lib.i
                include df0:include/graphics/graphics_lib.i

* MACRO-Definitionen


* Struktur für Windows

WINDOW          macro
                dc.w        \1          ;linke Ecke
                dc.w        \2          ;obere Ecke
                dc.w        \3          ;Breite
                dc.w        \4          ;Höhe
                dc.b        0,1         ;Farben
                dc.l        \5          ;Flags für Events
                dc.l     $1006          ;Window Flags
                dc.l        \6          ;Zeiger auf erstes Gadget
                dc.l        0           ;Check Mark
                dc.l        \7          ;Windowname
```

```
\8          dc.l      0         ;Screenpointer
            dc.l      0         ;Bit Map
            dc.w      \3        ;Mindest-Breite
            dc.w      \4        ;Mindest-Höhe
            dc.w      \3        ;maximale Breite
            dc.w      \4        ;maximale Höhe
            dc.w      15        ;Window-Typ
            endm

* Struktur für Screens

SCREEN          macro
          dc.w      0         ;linke Ecke
          dc.w      0         ;obere Ecke
          dc.w      \1        ;Breite
          dc.w      \2        ;Höhe
          dc.w      2         ;Tiefe (Bitplanes)
          dc.b      2,3       ;Farben
          dc.w      \3        ;Modus
          dc.w     15         ;Screentyp
          dc.l      0         ;Zeichensatz
          dc.l      \4        ;Screentitel
          dc.l      0         ;Gadgets
          dc.l      0         ;Bitmap
                endm


0000006E begin  lea intname,a1         ;Intuition-Library öffnen
00000074        clr.l d0
00000076        CALLEXEC OpenLibrary
0000007E        tst.l d0
00000080        beq fini2
00000084        move.l d0,_IntuitionBase

0000008A        lea grafname,a1        ;Grafik-Library öffnen
00000090        clr.l d0
00000092        CALLEXEC OpenLibrary
0000009A        tst.l d0
0000009C        beq fini3
000000A0        move.l d0,_GfxBase

000000A6        lea ffpname,a1         ;Mathematik-Library öffnen
000000AC        clr.l d0
000000AE        CALLEXEC OpenLibrary
000000B6        tst.l d0
000000B8        beq fini4
000000BC        move.l d0,_MathBase
```

```
000000C2        lea screen_defs,a0            ;Screen öffnen
000000C8        CALLINT OpenScreen
000000D2        move.l d0,screenpointer       ;Pointer für
000000D8        move.l d0,point1              ;Window

000000DE        lea plotwindow,a0             ;Window öffnen
000000E4        CALLINT OpenWindow
000000EE        move.l d0,windowpointer


000000F4        move.l screenpointer,a0       ;Adresse Bitmap
000000FA        move.l $58(a0),a0             ;holen
000000FE        move.l $08(a0),a4


00000102        move.l #255,d7                ;x-Radius
00000108        move.l #107,d6                ;y-Radius
0000010E        move.l #320,d5                ;x-Mittelpunkt
00000114        move.l #128,d4                ;y-Mittelpunkt

0000011A        move.l #26,zaehler
00000124        clr.b modus
0000012A        clr.l startwinkel             ;Startwinkel = 0 Grad
00000130        move.l #$C90FD943,endwinkel   ;Endwinkel =1.999 * PI

0000013A l1     move.l startwinkel,d2
00000140        move.l endwinkel,d3
00000146        bsr ellipse                   ;Ellipse setzen
0000014A        subi.l #10,d7
00000150        subq.l #1,zaehler
00000156        bne l1

0000015A        move.l #255,d7                ;x-Radius
00000160        move.l #27,zaehler

0000016A l2     move.l startwinkel,d2
00000170        move.l endwinkel,d3
00000176        bsr ellipse                   ;Ellipse setzen
0000017A        subq.l #4,d6
0000017C        subq.l #1,zaehler
00000182        bne l2


00000186        move.l #$FFFFF,d0             ;Warteschleife
0000018C loop   subq.l #1,d0
0000018E        bne loop
```

```
00000192          move.l windowpointer,a0          ;Window schließen
00000198          CALLINT CloseWindow

000001A2          move.l screenpointer,a0          ;Screen schließen
000001A8          CALLINT CloseScreen

000001B2 fini4    move.l _MathBase,a1
000001B8          CALLEXEC CloseLibrary
000001C0 fini3    move.l _GfxBase,a1
000001C6          CALLEXEC CloseLibrary
000001CE fini2    move.l _IntuitionBase,a1
000001D4          CALLEXEC CloseLibrary
000001DC fini1    rts

                  cnop 0,2

000001DE zaehler dc.l 0

000001E2 _IntuitionBase   dc.l 0
000001E6 _GfxBase         dc.l 0
000001EA _MathBase        dc.l 0

000001EE grafname        dc.b "graphics.library",0
000001FF                 cnop 0,2
00000200 intname         dc.b "intuition.library",0
                         cnop 0,2
00000212 ffpname         dc.b "mathffp.library",0
                         cnop 0,2


         * Strukturen
         * a) SCREENS

00000222 screen_defs     SCREEN 640,400,$8000,plottitel

00000242 plottitel   dc.b "Funktion plotten",0
00000253          cnop 0,4

00000254 screenpointer  dc.l 0

         * b) WINDOWS

00000258 plotwindow   WINDOW 0,0,640,256,$200,0,plotwindowname,point1

00000288 plotwindowname  dc.b "Kreis/Ellipsendarstellung",0
000002A2          cnop 0,4
```

```
000002A4 windowpointer dc.l 0
                          even


         *Ellipsenroutine*

         *Übergabe :          D7: X-Radius            D6:Y-Radius
         *(alle word)          D5: X-Mittelpunkt       D4: Y-Mittelpunkt
         *                     D2: Startwinkel (FFP)   D3: Endwinkel  (FFP)
         *                     A4: Zeiger auf Bitmap
         *                     modus : 0=setzen, 1=invertieren, 255=löschen
000002A8 ellipse          move.l d2,d0
000002AA                  bsr winkel              ;Startpunktberechnung
000002AE                  move.b d2,startqua      ;Startquadrant
000002B4                  move.w d0,startpunkt    ;Startpunkt
000002BA                  move.l d3,d0
000002BC                  bsr winkel              ;Endpunktberechnung
000002C0                  move.b d2,endqua        ;Endquadrant
000002C6                  move.w d0,endpunkt      ;Endpunkt
000002CC                  move.b startqua,d0
000002D2                  cmpi.b #1,d0            ;erster Quadrant?
000002D6                  bne ausschnitt          ;nein, nur Ausschnitt
000002DA                  move.w startpunkt,d0    ;Startpunkt = 0?
000002E0                  bne ausschnitt          ;nein, nur Ausschnitt
000002E4                  move.b endqua,d0
000002EA                  cmpi.b #4,d0            ;letzter Quadrant?
000002EE                  bne ausschnitt          ;nein, nur Ausschnitt
000002F2                  move.w endpunkt,d0
000002F8                  cmpi.w #390,d0          ;Endpunkt = 390?
000002FC                  bne ausschnitt          ;nein, nur Ausschnitt


* Ganzen Kreis oder Ellipse zeichnen (ca. 2.5-8 * schneller als DrawEllipse)


00000300 vollellipse      move.w #391,d3          ;Zähler setzen
00000304                  move.w #1,d2
00000308                  lea cos-1,a5            ;Zeiger auf SIN/COS- und
0000030E                  lea bittab,a2           ;Bit-Tabbellen
00000314 elloop           clr.l d0
00000316                  clr.l d1
00000318                  move.b (a5,d3.w),d0     ;Tabellenwert
0000031C                  mulu d7,d0              ;* x-Radius
0000031E                  divu #255,d0            ;/ 255
00000322                  move.w d0,xoff          ;= x-Offset
00000328                  move.b (a5,d2.w),d1     ;Tabellenwert
0000032C                  mulu d6,d1              ;* y-Radius
```

```
0000032E              divu #255,d1          ;/ 255
00000332              move.w d1,yoff        ;= y-Offset
00000338              add.w d5,d0           ;Mittelpunkts-
0000033A              move.w d0,xkoord      ;Koordinaten x
00000340              cmpi.w #639,d0        ;hinzuaddieren
00000344              bge notpunkt1         ;Bereichsüberschreitung
00000348              add.w d4,d1           ;prüfen
0000034A              move.w d1,ykoord
00000350              cmpi.w #255,d1
00000354              bge notpunkt1         ;das gleiche mit y
00000358              bsr plot              ;Punkt 1 = xm+xoff/ym+yoff
0000035C notpunkt1    move.w xkoord,d0
00000362              cmpi.w #639,d0
00000366              bge notpunkt2
0000036A              move.w d4,d1          ;wie oben für Punkt 2
0000036C              sub.w yoff,d1
00000372              move.w d1,ykoord1
00000378              blt notpunkt2
0000037C              bsr plot              ;Punkt 2 = xm+xoff/ym-yoff
00000380 notpunkt2    move.w d5,d0
00000382              sub.w xoff,d0
00000388              move.w d0,xkoord1
0000038E              blt notpunkt3         ;wie oben für Punkt 3
00000392              move.w ykoord,d1
00000398              cmpi.w #255,d1
0000039C              bge notpunkt3
000003A0              bsr plot              ;Punkt 3 = xm-xoff/ym+yoff
000003A4 notpunkt3    move.w xkoord1,d0
000003AA              blt notpunkt4
000003AE              move.w ykoord1,d1     ;wie oben für Punkt 4
000003B4              blt notpunkt4
000003B8              bsr plot              ;Punkt 4 = xm-xoff/ym-yoff
000003BC notpunkt4    addq.w #1,d2
000003BE              subq.w #1,d3
000003C0              bne elloop
000003C4              rts


         * Kreis- oder Ellipsenteil zeichnen

000003C6 ausschnitt   move.l #391,d0        ;Tabellen mit Werten
000003CC              moveq.l #1,d1         ;*Radius/255 aufbauen
000003CE              clr.l d2
000003D0              lea cos-1,a2
000003D6              lea austab1,a5
000003DC              lea austab2,a6
000003E2 ausloop1     clr.l d3
```

```
000003E4                 move.b (a2,d0.w),d3
000003E8                 mulu d7,d3
000003EA                 divu #255,d3
000003EE                 move.w d3,(a5,d2.w)
000003F2                 clr.l d3
000003F4                 move.b (a2,d1.w),d3
000003F8                 mulu d6,d3
000003FA                 divu #255,d3
000003FE                 move.w d3,(a6,d2.w)
00000402                 addq.w #1,d1
00000404                 addq.w #2,d2
00000406                 subq.w #1,d0
00000408                 bne ausloop1
0000040C                 lea bittab,a2
00000412                 move.w endpunkt,d0       ;Wenn Endpunkt = 0 oder 1,
00000418                 cmpi.w #1,d0
0000041C                 bhi weiter1              ;2 setzen,da sonst Fehler
00000420                 move.w #2,endpunkt
00000428 weiter1         clr.l d0
0000042A                 clr.l d1
0000042C                 move.b startqua,d0
00000432                 subi.b #1,d0
00000436                 mulu #16,d0              ;(Startquadrant-1) * 16
0000043A                 move.b endqua,d1         ;plus (Endquadrant-1) * 4
00000440                 subi.b #1,d1
00000444                 mulu #4,d1
00000448                 add.b d1,d0
0000044A                 addi.l #sprung,d0        ;Tabellenstart
00000450                 move.l d0,a0             ;Tabellenadresse
00000452                 move.l (a0),a0           ;Startadresse auslesen
00000454                 jmp (a0)                 ;und anspringen

00000456 sprung          dc.l s1e1,s1e2,s1e3,s1e4
00000466                 dc.l s2e1,s2e2,s2e3,s2e4
00000476                 dc.l s3e1,s3e2,s3e3,s3e4
00000486                 dc.l s4e1,s4e2,s4e3,s4e4

00000496                 cnop 0,4

00000498 austab1         ds.w 392
000007A8 austab2         ds.w 392


         * Start im 1., Ende im 1. Quadranten

00000AB8 s1e1            move.w startpunkt,d0
```

```
00000ABE                 cmp.w endpunkt,d0
00000AC4                 blt s1e11
00000AC8                 move.w #391,d1
00000ACC                 bsr erster
00000AD0                 move.w #1,d0
00000AD4                 bra s2e11
00000AD8 s1e11           move.w endpunkt,d1
00000ADE                 bra erster


         * Start im 1., Ende im 2. Quadranten

00000AE2 s1e2            move.w startpunkt,d0
00000AE8 s1e22           move.w #391,d1
00000AEC                 bsr erster
00000AF0                 move.w #1,d0
00000AF4                 move.w endpunkt,d1
00000AFA                 bra zweiter

         * Start im 1., Ende im 3. Quadranten

00000AFE s1e3            move.w startpunkt,d0
00000B04 s1e333          move.w #391,d1
00000B08                 bsr erster
00000B0C                 move.w #1,d0
00000B10 s1e33           move.w #391,d1
00000B14                 bsr zweiter
00000B18                 move.w #1,d0
00000B1C                 move.w endpunkt,d1
00000B22                 bra dritter


         * Start im 1., Ende im 4. Quadranten

00000B26 s1e4            move.w startpunkt,d0
00000B2C s1e4444         move.w #391,d1
00000B30                 bsr erster
00000B34                 move.w #1,d0
00000B38 s1e44           move.w #391,d1
00000B3C                 bsr zweiter
00000B40                 move.w #1,d0
00000B44 s1e444          move.w #391,d1
00000B48                 bsr dritter
00000B4C                 move.w #1,d0
00000B50                 move.w endpunkt,d1
00000B56                 bra vierter
```

```
        * Start im 2., Ende im 1. Quadranten

00000B5A s2e1           move.w startpunkt,d0
00000B60 s2e11          move.w #391,d1
00000B64                bsr zweiter
00000B68                move.w #1,d0
00000B6C s2e111         move.w #391,d1
00000B70                bsr dritter
00000B74                move.w #1,d0
00000B78 s2e1111        move.w #391,d1
00000B7C                bsr vierter
00000B80                move.w #1,d0
00000B84                move.w endpunkt,d1
00000B8A                bra erster


        * Start im 2., Ende im 2. Quadranten

00000B8E s2e2           move.w startpunkt,d0
00000B94                cmp.w endpunkt,d0
00000B9A                blt s2e22
00000B9E                move.w #391,d1
00000BA2                bsr zweiter
00000BA6                move.w #1,d0
00000BAA                bra s3e22
00000BAE s2e22          move.w endpunkt,d1
00000BB4                bra zweiter


        * Start im 2., Ende im 3. Quadranten

00000BB8 s2e3           move.w startpunkt,d0
00000BBE                bra s1e33


        * Start im 2., Ende im 4. Quadranten

00000BC2 s2e4           move.w startpunkt,d0
00000BC8                bra s1e44


        * Start im 3., Ende im 1. Quadranten

00000BCC s3e1           move.w startpunkt,d0
00000BD2                bra s2e111
```

```
         * Start im 3., Ende im 2. Quadranten

00000BD6 s3e2          move.w startpunkt,d0
00000BDC s3e22         move.w #391,d1
00000BE0               bsr dritter
00000BE4               move.w #1,d0
00000BE8 s3e222        move.w #391,d1
00000BEC               bsr vierter
00000BF0               move.w #1,d0
00000BF4               bra s1e22


         * Start im 3., Ende im 3. Quadranten

00000BF8 s3e3          move.w startpunkt,d0
00000BFE               cmp.w endpunkt,d0
00000C04               blt s3e33
00000C08               move.w #391,d1
00000C0C               bsr dritter
00000C10               move.w #1,d0
00000C14               bra s4e33
00000C18 s3e33         move.w endpunkt,d1
00000C1E               bra dritter


         * Start im 3., Ende im 4. Quadranten

00000C22 s3e4          move.w startpunkt,d0
00000C28               bra s1e444


         * Start im 4., Ende im 1. Quadranten

00000C2C s4e1          move.w startpunkt,d0
00000C32               bra s2e1111


         * Start im 4., Ende im 2. Quadranten

00000C36 s4e2          move.w startpunkt,d0
00000C3C               bra s3e222


         * Start im 4., Ende im 3. Quadranten

00000C40 s4e3          move.w startpunkt,d0
```

```
00000C46 s4e33          move.w #391,d1
00000C4A                bsr vierter
00000C4E                move.w #1,d0
00000C52                bra s1e333


         * Start im 4., Ende im 4. Quadranten

00000C56 s4e4           move.w startpunkt,d0
00000C5C                cmp.w endpunkt,d0
00000C62                blt s4e44
00000C66                move.w #391,d1
00000C6A                bsr vierter
00000C6E                move.w #1,d0
00000C72                bra s1e4444
00000C76 s4e44          move.w endpunkt,d1
00000C7C                bra vierter


         * Ersten Quadranten zeichnen *

00000C80 erster         move.w #391,d2          ;Zähler setzen und
00000C84                sub.w d0,d2
00000C86                move.w d2,zaehl1
00000C8C                move.w #391,d2          ;invertieren
00000C90                sub.w d1,d2
00000C92                move.w d2,zaehl2
00000C98 loopq1         move.w zaehl2,d2
00000C9E                lsl #1,d2               ;Zähler mal 2
00000CA0                move.w (a5,d2.w),d0
00000CA4                add.w d5,d0
00000CA6                cmpi.w #639,d0          ;x = xr + xoff
00000CAA                bge noeins
00000CAE                move.w (a6,d2.w),d3
00000CB2                move.w d4,d1
00000CB4                sub.w d3,d1             ;y = yr - yoff
00000CB6                blt noeins
00000CBA                bsr plot
00000CBE noeins         addq.w #1,zaehl2
00000CC4                move.w zaehl1,d0
00000CCA                cmp.w zaehl2,d0
00000CD0                bne loopq1
00000CD4                rts
```

```
        * Zweiten Quadranten zeichnen *

00000CD6 zweiter        move.w d1,zaehl1        ;Zähler setzen
00000CDC                move.w d0,zaehl2
00000CE2 loopq2         move.w zaehl2,d2
00000CE8                lsl #1,d2               ;Zähler mal 2
00000CEA                move.w (a5,d2.w),d0
00000CEE                add.w d5,d0
00000CF0                cmpi.w #639,d0          ;x = xr + xoff
00000CF4                bge nozwei
00000CF8                move.w (a6,d2.w),d1
00000CFC                add.w d4,d1             ;y = yr + yoff
00000CFE                cmpi.w #255,d1
00000D02                bge nozwei
00000D06                bsr plot
00000D0A nozwei         addq.w #1,zaehl2
00000D10                move.w zaehl1,d0
00000D16                cmp.w zaehl2,d0
00000D1C                bne loopq2
00000D20                rts


        * Dritten Quadranten zeichnen *

00000D22 dritter        move.w #391,d2          ;Zähler setzen und
00000D26                sub.w d0,d2
00000D28                move.w d2,zaehl1
00000D2E                move.w #391,d2          ;invertieren
00000D32                sub.w d1,d2
00000D34                move.w d2,zaehl2
00000D3A loopq3         move.w zaehl2,d2
00000D40                lsl #1,d2               ;Zähler mal 2
00000D42                move.w (a5,d2.w),d3
00000D46                move.w d5,d0
00000D48                sub.w d3,d0             ;x = xr - xoff
00000D4A                blt nodrei
00000D4E                move.w (a6,d2.w),d1
00000D52                add.w d4,d1             ;y = yr + yoff
00000D54                cmpi.w #255,d1
00000D58                bge nodrei
00000D5C                bsr plot
00000D60 nodrei         addq.w #1,zaehl2
00000D66                move.w zaehl1,d0
00000D6C                cmp.w zaehl2,d0
00000D72                bne loopq3
00000D76                rts
```

```
        * Vierten Quadranten zeichnen *

00000D78 vierter         move.w d1,zaehl1       ;Zähler setzen
00000D7E                 move.w d0,zaehl2
00000D84 loopq4          move.w zaehl2,d2
00000D8A                 lsl #1,d2              ;Zähler mal 2
00000D8C                 move.w (a5,d2.w),d3
00000D90                 move.w d5,d0
00000D92                 sub.w d3,d0
00000D94                 blt novier
00000D98                 move.w (a6,d2.w),d3
00000D9C                 move.w d4,d1
00000D9E                 sub.w d3,d1            ;y = yr - yoff
00000DA0                 blt novier
00000DA4                 bsr plot
00000DA8 novier          addq.w #1,zaehl2
00000DAE                 move.w zaehl1,d0
00000DB4                 cmp.w zaehl2,d0
00000DBA                 bne loopq4
00000DBE                 rts


00000DC0 xkoord          dc.w 0
00000DC2 xkoord1         dc.w 0
00000DC4 ykoord          dc.w 0
00000DC6 ykoord1         dc.w 0
00000DC8 xoff            dc.w 0
00000DCA yoff            dc.w 0
00000DCC startqua        dc.b 0
                         even
00000DCE endqua          dc.b 0
                         even
00000DD0 startpunkt      dc.w 0
00000DD2 endpunkt        dc.w 0
00000DD4 zaehl1          dc.w 0
00000DD6 zaehl2          dc.w 0
00000DD8 modus           dc.b 0
                         even
00000DDA startwinkel     dc.l 0
00000DDE endwinkel       dc.l 0


        * Quadranten- und Punktberechnung

00000DE2 winkel          move.l d0,a5           ;merken
00000DE4                 move.l #$80000043,d1   ;Ausdruck * 4
```

```
00000DEA               CALLFFP SPMul
00000DF4               move.l #$C90FDA43,d1    ;durch 2*PI dividieren
00000DFA               CALLFFP SPDiv
00000E04               CALLFFP SPFix           ;In Integer wandeln
00000E0E               move.l d0,a3            ;merken
00000E10               addi.b #1,d0            ;Plus 1
00000E14               move.b d0,d2            ;merken
00000E16               move.l a3,d0            ;Quadrant-1 ins FFP-Format
00000E18               CALLFFP SPFlt           ;wandeln
00000E22               move.l #$C90FDA41,d1    ;mit PI/2 multiplizieren
00000E28               CALLFFP SPMul
00000E32               move.l d0,d1            ;von Winkel abziehen
00000E34               move.l a5,d0
00000E36               CALLFFP SPSub
00000E40               move.l #$C3800049,d1    ;mit 391 multiplizieren
00000E46               CALLFFP SPMul
00000E50               move.l #$C90FDA41,d1    ;durch PI/2 dividieren
00000E56               CALLFFP SPDiv
00000E60               CALLFFP SPFix           ;in Integer wandeln
00000E6A               rts


          * Tabelle mit Cosinus- (vorwärts) und Sinus- (rückwärts) Werten


00000E6C cos      dc.b 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
00000E80          dc.b 20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35
00000E90          dc.b 36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
00000EA0          dc.b 52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67
00000EB0          dc.b 68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83
00000EC0          dc.b 84,85,86,87,88,89,90,91,92,93,94,94,95,96,97,98
00000ED0          dc.b 99,100,101,102,103,104,105,106,107,108,109,109,110
00000EDD          dc.b 111,112,113,114,115,116,117,118,119,120,120,121
00000EE9          dc.b 122,123,124,125,126,127,128,128,129,130,131,132
00000EF5          dc.b 133,134,135,135,136,137,138,139,140,141,141,142
00000F01          dc.b 143,144,145,146,147,147,148,149,150,151,152,152
00000F0D          dc.b 153,154,155,156,156,157,158,159,160,160,161,162
00000F19          dc.b 163,164,164,165,166,167,167,168,169,170,171,171
00000F25          dc.b 172,173,174,174,175,176,177,177,178,179,179,180
00000F31          dc.b 181,182,182,183,184,184,185,186,187,187,188,189
00000F3D          dc.b 189,190,191,191,192,193,193,194,195,195,196,197
00000F49          dc.b 197,198,199,199,200,200,201,202,202,203,204,204
00000F55          dc.b 205,205,206,207,207,208,208,209,210,210,211,211
00000F61          dc.b 212,212,213,214,214,215,215,216,216,217,217,218
00000F6D          dc.b 218,219,219,220,221,221,222,222,223,223,224,224
00000F79          dc.b 224,225,225,226,226,227,227,228,228,229,229,230
00000F85          dc.b 230,230,231,231,232,232,233,233,233,234,234,235
```

```
00000F91          dc.b 235,235,236,236,237,237,237,238,238,238,239,239
00000F9D          dc.b 239,240,240,240,241,241,241,242,242,242,243,243
00000FA9          dc.b 243,244,244,244,244,245,245,245,246,246,246,246
00000FB5          dc.b 247,247,247,247,248,248,248,248,249,249,249,249
00000FC1          dc.b 249,250,250,250,250,250,251,251,251,251,251,251
00000FCD          dc.b 252,252,252,252,252,252,252,253,253,253,253,253
00000FD9          dc.b 253,253,253,253,254,254,254,254,254,254,254,254
00000FE5          dc.b 254,254,254,254,254,254,254,254,254,254,254,254
00000FF1          dc.b 254,254


         * Punkt setzen/löschen/invertiern

00000FF4 plot     and.l #$FFFF,d0          ;INT(X/8)*8 = Byte in Zeile
00000FFA          move.w d0,a1
00000FFC          divu #8,d0
00001000          and.l #$FF,d1            ;y*80 = Zeilenoffset
00001006          mulu #80,d1
0000100A          add.l a4,d1              ;Start Bitmap
0000100C          add.w d0,d1
0000100E          move.l a1,d0
00001010          move.b (a2,d0.w),d0      ;Bit holen
00001014          move.l d1,a1
00001016          move.b (a1),d1           ;Grafikbyte holen
00001018          tst.b modus              ;Modus holen
0000101E          beq setz                 ;Punkt setzen
00001022          bmi lösch                ;Punkt löschen

         * Punkt invertieren

00001026          eor.b d0,d1              ;Bit invertieren
00001028          move.b d1,(a1)           ;Byte setzen
0000102A          rts

         * Punkt setzen

0000102C setz     or.b d0,d1               ;Bit setzen
0000102E          move.b d1,(a1)           ;Byte setzen
00001030          rts

         * Punkt löschen

00001032 lösch    eor.b #$FF,d0            ;Byte umdrehen
00001036          and.b d0,d1              ;Bit löschen
00001038          move.b d1,(a1)           ;Byte setzen
0000103A          rts
```

```
        * Bits der x-Koordinaten

0000103C bittab          dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000104C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000105C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000106C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000107C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000108C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000109C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010AC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010BC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010CC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010DC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010EC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000010FC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000110C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000111C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000112C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000113C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000114C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000115C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000116C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000117C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000118C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000119C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011AC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011BC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011CC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011DC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011EC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000011FC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000120C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000121C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000122C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000123C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000124C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000125C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000126C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000127C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000128C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
0000129C                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
000012AC                 dc.b 128,64,32,16,8,4,2,1,128,64,32,16,8,4,2,1
```

Die eigentliche Ellipsenroutine beginnt beim Label »ellipse« und wird von dem Hauptprogramm wiederholt aufgerufen. Dabei müssen folgende Parameter übergeben werden:

D2 : Startwinkel im FFP-Format und Bogenmaß (0 < D2 < 2*PI)
D3 : Endwinkel im FFP-Format und Bogenmaß (0 < D3 < 2*PI)
D4 : y-Koordinate Mittelpunkt (0 < D4 < 256 (512 im Interlace-Modus))
D5 : x-Koordinate Mittelpunkt (0 < D5 < 320 (640 im HiRes-Modus)
D6 : y-Radius (0 < D6 < 256)
D7 : x-Radius (0 < D7 < 256)
A4 : Zeiger auf Bitmap (Langwort)
modus : Zeichenmodus (Byte): 0=Setzen, 1=Invertieren, 255=Löschen

Es ist wichtig zu beachten, daß die Winkel maximal nur bis zu dem Wert 1.999*PI zulässig sind. Im FFP-Format wird diese Zahl als $C90FD943 dargestellt. Die Register D4 bis D7 werden innerhalb der Routine nicht verändert, so daß Sie die Radien und die Mittelpunktskoordinaten nicht unbedingt vor jedem Aufruf neu setzen müssen. Die Register D2 und D3 werden hingegen modifiziert, womit Sie die Winkel also jedesmal neu festlegen müssen.

Da unser Programm nicht mit Winkeln, sondern mit Integer-Werten rechnet, muß man diese zunächst entsprechend umrechnen. Die Eingabemöglichkeit im Bogenmaß dient allein dazu, den Bedienungskomfort für den Anwender zu erhöhen, unser Programm kann mit den FFP-Werten jedoch nichts anfangen, waren wir doch froh, überhaupt ohne Fließkommazahlen auskommen zu können!

Wir benötigen für unseren Rechenalgorithmus vielmehr den Quadranten und den zugehörigen Punkt innerhalb des Quadranten (0–391), um unsere Integer-Schleife dort ansetzen zu können. Den Quadranten kann man nach der Formel

```
Quadrant = INT(Winkel*(4/(2*PI)))+1
```

berechnen. Wenn Sie z.B. als Winkel 100 Grad (=0.8726*PI) eingeben, wird nach unserer Formel INT(0.8726*4/2)+1 der 2. Quadrant berechnet, was sicherlich richtig ist, da dieser von 90–180 Grad geht. Im Prinzip handelt es sich um eine einfache Verhältnisgleichung, wobei unser Winkel mit dem einer Vollellipse (2*PI) ins Verhältnis gesetzt wird. Das Ergebnis, welches immer kleiner als eins sein muß, wird dann mit der vorhandenen Anzahl der Quadranten multipliziert. Nachdem der Quadrant gefunden wurde, muß nur noch der Start- bzw. Endpunkt unserer Integer-Schleife gefunden werden. Wenn wir diese nämlich immer von 0 bis 391 laufen ließen, würde immer eine komplette Viertelellipse gezeichnet werden. Hierzu kann man folgende Formel verwenden:

```
Wert = INT(Winkel - ((Quadrant-1)*(PI/2)))*(391/(PI/2))
```

Obwohl dieses Zahlenwerk mehr verwirrend als sinnvoll aussieht, ist es durchaus erklärbar: Um das Verhältnis des innerhalb des Quadranten überstrichenen Winkels zu 90 Grad (maximaler Winkel eines Quadranten) zu finden, muß zunächst der Gesamtwinkel auf einen Anteil reduziert werden, der zwischen 0 und 90 Grad liegt. Dafür wird 90 Grad (=PI/2) vom Gesamtwinkel einmal weniger abgezogen als der Wert des Quadranten, z.B.

bei 100 Grad (2.Quadrant) werden einmal 90 Grad subtrahiert, so daß das Ergebnis 10 Grad lautet. Dies ist der Anteil des Winkels, der seit Eintritt in den 2. Quadranten (bei 90 Grad) überstrichen worden ist. Jetzt wird wiederum eine Verhältnisgleichung aufgestellt: Da eine Viertelellipse durch einen Schleifenwert von 391 erzeugt wird, wird ein beliebiger kleinerer Ausschnitt durch diesen Wert multipliziert mal dem Verhältnis von Winkel zu 90 Grad repräsentiert, z.B. 10 Grad durch den Wert 391*(10/90) = 43. Damit hat man den zugehörigen Quadranten und Schleifenwert berechnet, unser Programm durchläuft die Routine einmal für den Start- und einmal für den Endwinkel.

Als nächstes wird geprüft, ob eine Vollellipse gezeichnet werden soll. Diesen Sonderfall erkennt man daran, daß als Startpunkt der erste Quadrant mit dem Schleifenwert 0 und als Endpunkt der vierte Quadrant mit dem Schleifenwert 391 berechnet wurde. In diesem Fall wird in die superschnelle Routine gesprungen, die nach oben erläutertem Verfahren pro Schleifendurchlauf vier Punkte setzt.

Sehr viel komplizierter ist das Zeichnen von Ellipsenausschnitten. Hierfür mußten getrennte Zeichenroutinen für jeden Quadranten implementiert werden, die zwar nach gleichem Verfahren arbeiten, aber pro Schleifendurchlauf nur einen Punkt setzen. Das Problem besteht darin, daß insgesamt 16 verschiedene Quadrantenkombinationen existieren.

Jede dieser Kombinationen benötigt eine eigene Routine, die nacheinander die Zeichenroutinen der einzelnen Quadranten aufruft. Die Startadresse der zugehörigen Routine wird aus einer Tabelle ausgelesen. Der Index für diese Tabelle wird berechnet, indem man den Anfangsquadranten mal 4 nimmt und den Endquadranten hinzuaddiert. Dadurch sind Werte von 5 bis 20 möglich, wobei jede Quadrantenkombination eindeutig einem dieser Werte zugeordnet ist. Da die Adressen in Langwörtern abgelegt sind, müssen diese Indexe noch mit 4 multipliziert werden, bevor die Sprungadresse ausgelesen werden kann.

Startquadrant	Endquadrant	Startquadrant	Endquadrant
1	1	3	1
1	2	3	2
1	3	3	3
1	4	3	4
2	1	4	1
2	2	4	2
2	3	4	3
2	4	4	4

Als Beispiel möchte ich die Routine behandeln, die den Ausschnitt vom ersten in den dritten Quadranten zeichnet (»s1e3«): Zunächst muß der erste Quadrant betrachtet werden. Hier ist genau der Abschnitt vom Start (Startpunkt) bis zum Ende (391) zu zeichnen. Der zweite Quadrant muß komplett aufgebaut werden (Start=1, Ende =391), während der dritte Quadrant genau vom Anfang (1) bis zum berechneten Endpunkt gezeichnet werden muß.

Genauso kann man sich die Funktion der übrigen 15 Kombinationen verdeutlichen. Ich kann Ihnen jetzt nur noch viel Spaß bei der Verwendung dieser Routine wünschen, die sicherlich in keinem Grafikprogramm fehlen sollte.

4.8 Das IFF-Bild-Format

Die Umsteiger vom C64 unter Ihnen, liebe Leser, können ein Lied davon singen, was es auf diesem Computer bedeutete, verschiedene Grafikprogramme zu verwenden. Solange man mit jedem Programm nur die von ihm erzeugten Bilder verarbeiten wollte, traten keine Probleme auf, aber wehe, Sie wollten ein Bild des Programms A mit dem Programm B weiterverarbeiten! Die harmloseste Konsequenz bestand darin, daß der Bildschirm mit unsinnigen Daten gefüllt wurde, Systemabstürze konnte man aber auch vermelden, wenn z.B. ein Programm die Daten nicht analysieren konnte und dann in einer Endlosschleife verharrte. Der Grund für diese Zwischenfälle liegt in der unterschiedlichen Art und Weise, in der die Grafikdaten von den einzelnen Programmen abgespeichert wurden. Man möchte annehmen, daß es doch keine großen Unterschiede geben könnte, schließlich mußten alle Programme die Grafikbitplane irgendwie auf der Magnetscheibe verewigen. Die Phantasie der Programmierer kannte jedoch keine Grenzen, wenn es darum ging, die erzeugten Files möglichst kurz zu halten. Dazu wurden Datenkompressoren aller Art eingesetzt, die zwar alle das gleiche erreichten (kürzere Files), das Grafikfile für andere Programme jedoch unlesbar machten. Die logische Konsequenz bestand darin, daß in einem Fachmagazin Dutzende von Konvertierungsprogrammen erschienen, mit denen man das Grafikfile des einen in ein anderes Format umwandeln konnte. Leider dauerte dieser Konvertierungsvorgang so lange, daß man bequem mehrere unkomprimierte Grafiken in dieser Zeit hätte einlesen können, weshalb man Grund hatte, etwas frustriert zu sein.

Die Firma Commodore war ebenfalls frustriert und beschloß, so einen Krampf (anders kann man die C64-Zustände nicht beschreiben) auf dem Amiga nicht zuzulassen. Sie entwickelte darum zusammen mit einer bekannten Softwarefirma ein Standardformat, das sich neben Grafiken auch auf andere Files aller Art bezieht, das sogenannte IFF (Interchange-File-Format).

Bedauerlicherweise setzten sich aber gewisse Programme, z.B. DeluxePaint, über dieses Standard-Format hinweg (nach dem Motto: Ich bin sowieso der Beste, wer sollte Grafiken von mir in anderen primitiven Zeichenprogrammen verwenden ...?), so daß man diese Grafiken erst wieder konvertieren muß, um das IFF-Format zu erhalten. Zum Glück gibt es aber genug ernstzunehmende Programme (das bekannteste ist »Grafikcraft«), die Grafikfiles in genau diesem Format erstellen.

Das IFF-Format besteht aus einzelnen Teilen, die durch einen 4stelligen Titel angezeigt werden. Nach jedem Titel folgt in einer festgelegten Reihenfolge eine bestimmte Anzahl von Datensätzen, aus denen man z.B. die Größe der BitMaps erkennen kann. Es ist jedoch nicht zwangsläufig erforderlich, alle Teile abzuspeichern, einige können sogar doppelt auftreten. Zunächst kommt der Kopf der Grafik:

vier Bytes: »FORM«
vier Bytes: Größe der Datei in Bytes

Durch das nun folgende Codewort wird der Start der eigentlichen Daten angezeigt:

vier Bytes: »ILBM«

Jetzt kommen die eigentlichen Daten. Dafür stehen acht verschiedene Titel zur Verfügung:

DPPV : GraphiCraft-spezifisch
BMHD : BitMap-Header (Grafikinformation)
CMHD : GraphiCraft-spezifisch
CMAP : Color-Map (Farbinformation)
CRNG : Color-Range (Farbverläufe)
BODY : Bilddaten der Grafik
CAMG : ViewPort-Mode des Screens
CCRT : CycleInfo (Information zur Farbanimation)

Wie oben schon erwähnt, ist es nicht erforderlich, alle Titel abzufragen. Die wichtigsten Dinge stehen in den Titeln CMAP, BODY und BMHD, da sie die Grundinformationen der Grafik enthalten. Falls Sie einmal in die Verlegenheit kommen sollten, eine Grafik einzulesen, die z.B. auch den Titel CAMG enthält, können Sie diesen getrost überlesen. Nun aber zum Aufbau der wichtigen drei Datenblöcke:

BitMapHeader

vier Bytes : »BMHD«
vier Bytes : Wert 20 (Länge von folgenden Daten)
zwei Bytes : Breite des Screens
zwei Bytes : Höhe des Screens
vier Bytes : Wert 0
ein Byte : Tiefe des Screens
fünf Bytes : Wert 0
ein Byte : Wert 10
ein Byte : Wert 11
zwei Bytes : Breite des Screens
zwei Bytes : Höhe des Screens

ColorMap

vier Bytes : »CMAP«
vier Bytes : Anzahl der Farben mal 3
anschließend : für jede Farbe je 3 Byte (Rot, Grün, Blau)

Body

vier Bytes : »BODY«
vier Bytes : Größe der Grafik = Höhe*Breite*Tiefe/8
anschließend : Grafik, reihenweise gespeichert (erste Zeile 1. Bitplane,
erste Zeile 2. Bitplane, usw.)

Die praktische Anwendung bekommen wir in dem Beispielprogramm im nächsten Kapitel, das es erlaubt, IFF-Grafiken mit Hilfe einer komfortablen Fileauswahl zu laden und zu speichern.

Ein-Ausgabeoperationen durch Amiga-DOS

Kapitel 5

Ohne Ein-/Ausgabeoperationen läuft normalerweise bei einem Computer überhaupt nichts. Es ist deshalb erstaunlich, daß wir erst jetzt zu diesem Thema kommen. Nun, der wichtigste Grund, weshalb ich Ihnen bislang die Informationen über das Amiga-DOS verschwiegen habe, liegt in Intuition. Dort haben wir z.B. mit den Gadgets Möglichkeiten gefunden, Eingaben ohne das DOS zu tätigen, ein spezielles Merkmal des Amiga-Betriebssystems. Genauso verhielt es sich mit der Ausgabe: Eine Funktion mit dem Namen PrintI-Text erlaubte es uns, Ausgaben auf dem Bildschirm zu tätigen, ohne die DOS-Library überhaupt einladen zu müssen.

Erst wenn es um die Ansteuerung des Druckers (Wertetabelle) oder des Disketten-Laufwerks ging (Chart-Grafik), mußten wir einige DOS-Funktionen benutzen. Theoretisch kann man aber auf dem Amiga Programme entwickeln, die keine einzige DOS-Funktion benötigen, wie z.B. unsere Ellipsenroutine. Diese Tatsache ist in der Computerwelt keineswegs selbstverständlich, so konnte man auch beim C64 gar nicht daran denken, ein Programm ohne diese zu schreiben, das die Kernal-Funktionen ignorieren würde.

Neben den Sekundärgeräten Drucker (PRT:), Diskettenlaufwerk und Festplatte (DFx:) ist das Amiga-DOS noch für die Steuerung der seriellen (SER:) und parallelen (PAR:) Schnittstelle zuständig. Dazu kommt noch das logische Gerät mit dem Namen Console (CON:) und die RAM-Disk (RAM:).

Wir werden uns hier auf die für die tägliche Anwendung wichtigsten Einheiten PRT:, DFx: und CON: beschränken. Da der prinzipielle Programmiervorgang sowieso immer gleich ist, können Sie diese Informationen leicht auch auf die anderen logischen Geräte übertragen.

5.1 Allgemeine DOS-Funktionen

Grundlage der Programmierung des Amiga-DOS stellt die DOS-Library dar. Sie enthält neben den universell zu verwendenden Routinen für z.B. das Öffnen und Schließen einer Datei auch spezielle Funktionen vornehmlich für Diskettenlaufwerke und Festplatten. Zunächst zu denjenigen Funktionen, die von allen Einheiten unbedingt benötigt werden.

Datei öffnen: **Open**

Um eine Ein- oder Ausgabe tätigen zu können, muß man zunächst eine Datei öffnen. Das wichtigste Merkmal ist dabei der Dateiname, der aus der logischen Einheit und zusätzlich z.B. bei Diskettenfiles aus dem Filenamen besteht, wenn der Name der Einheit allein nicht ausreicht, um eine eindeutige Identifizierung zu ermöglichen (so würden z.B. durch den Dateinamen DF0: alle Files auf der Diskette im Laufwerk 0 angesprochen, während für die Parallelschnittstelle der Dateiname PAR: genügt, weil nur eine Schnittstelle vorhanden ist).

Außerdem muß beim Aufruf der Open-Funktion der sogenannte Open-Modus übergeben werden, der nichts anderes aussagt, als ob die angesprochene Datei schon existiert (zum Lesen oder Schreiben, Modus = 1005) oder ob sie neu angelegt werden soll (nur zum Schreiben, Modus = 1006). Auf einigen Geräten wie dem Drucker kann man natürlich Daten nur ausgeben, so daß hierfür der Lese-Modus nicht verwendet werden kann. Als Ergebnis erhält man einen Zeiger auf die FileHandle-Struktur, den wir unbedingt sichern müssen, da er für die folgenden Operationen von Bedeutung ist. Der Inhalt dieser Struktur selbst ist aber zunächst uninteressant.

```
move.l #dateiname,d1     ;Zeiger auf Dateinamen
move.l #1006,d2          ;Modus :Neu
CALLDOS Open             ;Datei öffnen
move.l d0,filehandle     ;Pointer sichern

dateiname  dc.b "PRT:",0    ;Drucker
filehandle dc.l 0
```

Falls als Ergebnis eine Null zurückkommt, konnte die Datei nicht geöffnet werden.

Daten schreiben: **Write**

Durch diese Funktion können wir Daten in die geöffnete Datei hineinschreiben, wobei besonders auffällt, daß dies nicht immer nur ein Zeichen, sondern theoretisch beliebig viele sein können. Damit kann man sehr viel Programmierarbeit sparen. Übergeben werden ein Zeiger auf den Puffer, in dem unsere Daten stehen, sowie die Anzahl der Bytes, die wir beabsichtigen, in die Datei zu schreiben. Als Ergebnis bekommen wir die Zahl der Bytes zurück, die geschrieben werden konnten. Bei einem Fehler kommt eine -1 zurück. Wenn die Datei bereits existierte und damit schon Daten enthielt, werden die neuen Daten angehängt, falls die Datei mit dem Modus »Alt« geöffnet wurde. Wenn Sie jedoch (wenn auch versehentlich) den Modus »Neu« gesetzt haben, kennt das DOS keine Gnade: Die alten Daten werden rücksichtslos überschrieben.

```
move.l filehandle,d1     ;Dateipointer holen
move.l #puffer,d2        ;Anfang des Datenpuffers
move.l #zahl,d3          ;Zahl der zu schreibenen Bytes
CALLDOS Write            ;Daten schreiben
```

Daten lesen: **Read**
Die Read-Funktion arbeitet analog zu der vorhergehenden Write-Funktion, funktioniert aber logischerweise nur dann, wenn Sie die Datei mit dem Modus »Alt« geöffnet haben. Die Parameter und das Ergebnis sind identisch.

```
move.l filehandle,d1     ;Dateipointer holen
move.l #puffer,d2        ;Anfang des Datenpuffers
move.l #zahl,d3          ;Zahl der zu schreibenen Bytes
CALLDOS Read             ;Daten lesen
```

Datei schließen : **Close**
Sie sollten sich unbedingt angewöhnen, sofort wenn die Datei nicht mehr benötigt wird, diese wieder zu schließen, damit eventuelle andere Tasks darauf zugreifen können (wichtig besonders bei Unikaten wie Drucker). Besonders, wenn man von Ein-Task-Rechnern wie dem C64 umgestiegen ist, wo immer nur das eigene Programm auf eine Einheit zugreifen konnte und wo es deshalb üblich war, eine Datei am Anfang des Programms zu öffnen und am Ende wieder zu schließen, sollte man sich das Multitasking-Konzept des Amiga immer wieder vergegenwärtigen !

```
move.l filehandle,d1     ;Dateipointer holen
CALLDOS Close            ;Datei schließen
```

5.2 Spezielle Funktionen für Diskettenlaufwerke

Die obigen vier Funktionen reichen für einen normalen Dateizugriff vollkommen aus. Bei Disketten sieht die Sache aber etwas anders aus: Hier wünscht man sich weitere Informationen über Files, Unterdirektorys, etc., möchte Files löschen, umbenennen oder vorm Löschen schützen. Dazu stellt uns die DOS-Library weitere Funktionen zur Verfügung.

»Schlüssel« zur Datei ermitteln: **Lock**
Um die folgenden Funktionen ausführen zu können, muß zunächst der Schlüssel zu dem betreffenden File oder Direktory ermittelt werden. Bildlich kann man es sich in etwa so vorstellen, daß das DOS versucht, den Pfad durch Direktorys und Unterdirektorys zu finden. Als Ergebnis bekommt man einen Zeiger auf die Lock-Struktur des Files oder Direktorys, den die folgenden Funktionen noch benötigen werden. Falls die Datei nicht vorhanden ist, bekommt man eine Null zurück. Übergeben wird ein Zeiger auf den Dateinamen sowie die Angabe des Modus (-1: Datei kann nur beschrieben werden, -2: Datei kann gelesen werden).

```
move.l #name,d1          ;Zeiger auf Dateinamen
move.l #-2,d2            ;Modus "Lesen"
CALLDOS Lock             ;Schlüssel ermitteln
move.l d0,locksave       ;uns sichern
```

```
locksave   dc.l 0
name    dc.b "DF0:",0     ;Schlüssel zum internen Diskettenlaufwerk ermitteln
                          ;Alle Files und Direktorys des Hauptdirektorys
                          ;werden gefunden ("dir1 (DIR)","dir2 (DIR)","file1")
        dc.b "dir1",0     ;Schlüssel zum Direktory "dir1" ermitteln
                          ;Alle Files und Unterdirektorys des Direktorys
                          ;"dir1" werden gefunden ("dir3 (DIR)","file2")
        dc.b "dir2/c",0   ;Schlüssel zum File "c" im Direktory "dir2"
                          ;ermitteln
                          ;Nur das File "c" im Direktory "dir2" wird gefunden
        dc.b "dir2/c.*",0;Schlüssel zu den Files, die mit "c" beginnen,
                          ;ermitteln
                          ;Die Files "c", "copy" und "cancel" werden gefunden
```

Ersten Eintrag ermitteln: **Examine**

Durch diese Funktion wird der erste Eintrag ermittelt, auf den der mit der Lock-Funktion gefundene Schlüssel paßt. Falls man dort z.B. mit »DF0:« ein ganzes Laufwerk angesprochen hat, werden Informationen über die eingelegte Diskette ausgegeben, nicht über ein File!

Die Informationen werden in den sogenannten File-Info-Block geschrieben, dessen Zeiger Sie der Routine übergeben müssen. Wichtig hierbei ist, daß dieser Speicherbereich bei einer Langwort-Adresse beginnt, da die Routine sonst falsche Informationen in den Puffer hineinschreibt. Der Info-Block besitzt eine Länge von 260 Byte und ist wie folgt aufgebaut.

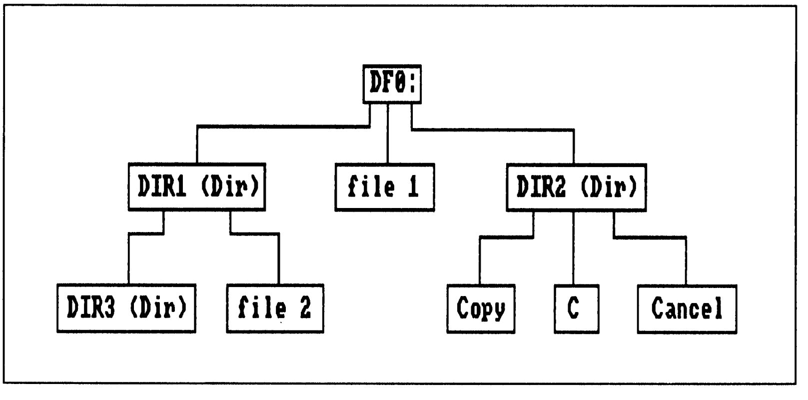


Bild 5.1: Beispiele für zulässige Dateinamen

Offset	Name	Typ	Bedeutung
0	DiskKey	L	Diskettennummer
4	DirEntryType	L	Eintragstyp (>0: Direktory, <0: Datei)
8-115	Filename		Dateiname (mit 0-Byte abgeschlossen)
116	Protection	L	Code-Langwort für z.B. Löschschutz (s.u.)
120	EntryType	L	Eintragstyp
124	Size	L	Dateilänge in Bytes
128	NumBlocks	L	Anzahl der belegten Diskettenblocks
132	Days	L	Erstellungsdatum
136	Minute	L	Erstellungszeit (Minuten)
140	Tick	L	Erstellungszeit (Sekunden)
144-259	Comment		Kommentar (116 Byte)

Wenn Sie ein Laufwerk untersuchen, steht ab der Adresse 8 der Diskettenname zur Verfügung.

```
move.l locksave,d1        ;Schlüssel von Lock-Funktion
move.l #fileinfo,d2       ;Zeiger auf File-Info-Block
CALLDOS Examine           ;Suchen

          cnop 0,4        ;Adresse auf Langwort
fileinfo  ds.b 260        ;Platz für 260 Byte
```

Folgeeinträge ermitteln: **ExNext**
Meistens paßt der gefundene Schlüssel nicht nur auf ein File, sondern auf mehrere, so z.B. der eines ganzen Laufwerks auf die mit Examine untersuchte Diskette wie auch auf alle Directorys und Files (siehe oben).

Die ExNext-Funktion ermittelt nun den folgenden Eintrag auf den mit der Examine-Funktion gefundenen. Durch weitere Aufrufe dieser Funktion können solange alle passenden Einträge ermittelt werden, bis als Ergebnis eine Null zurückkommt. Dann wurde der letzte Eintrag gefunden, zu dem der mit Lock ermittelte Schlüssel paßt. Bedeutsam ist diese Funktion im Zusamnmenspiel mit Lock und Examine besonders für die Directory-Ausgabe oder für eine Fileauswahl, wie wir sie im folgenden Beispielprogramm kennenlernen werden.

```
loop move.l locksave,d1        ;Schlüssel von Lock-Funktion
     move.l #fileinfo,d2       ;Zeiger auf File-Info-Block
     CALLDOS ExNext            ;Suchen
     tst.l d0                  ;Weitersuchen, bis letzter
     bne loop                  ;Eintrag gefunden wurde
```

Oberdirectory ermitteln: **ParentDir**
Mit Hilfe dieser Funktion ist es möglich, festzustellen, in welchem Verzeichnis sich eine bestimmte Datei oder ein Unterdirectory befindet. Als Ergebnis erhält man einen Zeiger auf die Lock-Struktur des Oberverzeichnisses, übergeben wird der Zeiger auf die Lock-Struktur der jeweiligen Datei.

```
move.l locksuch,d1          ;Zeiger auf Lock-Struktur Datei/Directory
CALLDOS ParentDir
move.l d0,lockoberdir       ;Zeiger auf Lock-Struktur Oberverzeichnis

locksuch      dc.l 0
lockoberdir   dc.l 0
```

Diese Funktion ist immer dann von Bedeutung, wenn man die Lock-Struktur eines Eintrages nicht kennt, aber weiß, daß er sich im selben Directory befindet wie ein anderer, bekannter Eintrag. Man kann dann das Oberdirectory ermitteln und anschließend mit den Funktionen Examine und ExNext die Lock-Struktur des gesuchten Eintrages ermitteln. Wenn man sich Bild 5.1 ansieht, kann man z.B. folgenden Fall konstruieren: Angenommen, Sie kennen die Lock-Struktur des Files »c« im Unterdirectory »dir2«. Sie möchten aber gerne wissen, welche anderen Files ebenfalls in diesem Directory vorkommen. Die Anwendung der Lock-Funktion auf das File »c« würde zu keinem Ergebnis führen, da es sich hierbei schon um die unterste Stufe, ein File, handelt. Wenn man aber die Lock-Struktur des Directorys »dir2« kennen würde, könnte man durch Anwendung der Examine- und ExNext-Funktionen die Lock-Strukturen aller in diesem Directory gespeicherten Files erhalten, also neben dem File »c« selbst auch die von »copy« und »cancel«.

Neues Directory setzen: **CurrentDir**
Die Anwendung dieser Funktion ist dann notwendig, wenn man bei der Untersuchung mit den obigen Funktionen auf ein Unterdirectory stößt und dessen Inhalt analysieren möchte. Eine Lock-Analyse ausgehend vom Hauptdirectory in der Struktur in Bild 5.1 z.B. würde zwar die Schlüssel für die Unterdirectorys »dir1«, »dir2« und »dir3« liefern, der Inhalt dieser würde aber zunächst verborgen bleiben. Erst das Neusetzen des Directorys auf eines dieser Unterdirektories würde den Inhalt desselben zugänglich machen. Freundlicherweise liefert die CurrentDir-Funktion nach der Übergabe der Lock-Struktur des gewünschten Directorys diejenige des letzten Directorys zurück, so daß man nach getaner Arbeit das aktuelle Directory auf den alten Wert zurücksetzen kann. Zum Programmende sollte man dies spätestens durchführen, damit nicht ein frustrierter Anwender zurückbleibt, der vergeblich versucht, auf irgendwelche Files zuzugreifen, die durch die CurrentDir-Funktion unerreichbar geworden sind.

```
move.l locksave,d1      ;Zeiger auf Lock-Struktur des neuen Directorys
CALLDOS CurrentDir      ;als aktuelles Directory setzen
move.l d0,altlock       ;Zeiger auf Lock-Struktur des alten Directorys

altlock  dc.l 0
```

Zum Programmende (spätestens):

```
move.l altlock,d1       ;Zeiger auf Lock-Struktur des alten Directorys
CALLDOS CurrentDir      ;wieder als aktuelles Directory setzen
```

Datei löschen: **DeleteFile**
Die Funktion entspricht der des CLI-Befehls Delete. Man kann eine Datei oder ein Directory löschen, letzteres aber nur dann, wenn es keine weiteren Files oder Direktories mehr enthält. Als Übergabeparameter fungiert einmal nicht ein Zeiger auf die Lock-Struktur, sondern auf den File- oder Directorynamen selbst. Falls nicht gelöscht werden konnte, wird eine Null zurückgegeben.

```
move.l #filename,d1     ;Zeiger auf Filenamen
CALLDOS DeleteFile      ;File löschen

filename dc.b"DF0:dir2/copy",0 ;File "copy" im Unterdirectory "dir1" löschen
```

Datei umbenennen: **RenameFile**
Funktioniert analog zum CLI-Befehl Rename. Es werden Zeiger auf den alten sowie den neuen Dateinamen übergeben. Falls schon ein File mit dem neuen Namen existiert, wird nicht umbenannt. Auch hier zeigt eine Null als Rückgabewert an, das die Operation nicht geklappt hat.

```
move.l #filealt,d1      ;Zeiger auf alten Filenamen
move.l #fileneu,d2      ;Zeiger auf neuen Filenamen
CALLDOS RenameFile      ;File umbenennen

filealt   dc.b "DF0:dir2/copy",0    ;Es wird vergeblich versucht, "copy" in
fileneu   dc.b "DF0:dir2/cancel",0  ;in "cancel" umzubenennen
```

Letzten Fehler holen: **IOErr**
Falls eine Funktion nicht erfolgreich beendet wurde, wird einem dies einfach durch den Rückgabewert Null mitgeteilt. In eigenen Programmen ist es aber erforderlich, detaillierte Meldungen an den Benutzer auszugeben (es sei denn, es sind Beispielprogramme in einem Buch <Schluck>...), damit dieser entsprechend darauf reagieren kann. Die IOErr-Funktion liefert eine Fehlernummer zurück, deren Bedeutung im Anhang erklärt ist.

```
CALLDOS  IOErr         ;Fehlercode holen
move.l d0,error        ;und sichern

dc.l error
```

Schützen einer Datei oder eines Directorys: **SetProtection**
Hiermit ist es möglich, eine Datei oder ein Directory vor bestimmten Operationen zu schützen. Dafür wird der Routine ein Langwort übergeben, in dem die Bits 0 bis 3 angeben, vor welcher Aktion geschützt werden soll:

gesetztes Bit	Wert	Bedeutung
0	1	Datei ist nicht löschbar
1	2	Datei ist nicht ausführbar
2	4	Datei ist nicht überschreibbar
3	8	Datei ist nicht lesbar

Besonders bedeutsam ist das Schützen vor der Ausführung: Stellen Sie sich vor, Sie haben eine Adreßdatei angelegt und ein unwissender Mitbenutzer Ihres Amiga versucht nun, diese Datei zu starten! Die Datensätze würden als Assemblercodes interpretiert, man kann guten Gewissens sein Hab und Gut darauf verwetten, daß es sich der Guru nicht nehmen lassen wird, uns mit seiner Anwesenheit zu erfreuen!

```
move.l #dateiname,d1      ;Zeiger auf Datei- oder Directoryname
move.l #code,d2           ;Schutzmaske
CALLDOS SetProtection     ;File schützen

dateiname  dc.b "DF0:dir2/copy",0
code    dc.l %00000000000000000000000000001001 ;Datei unlesbar und unlöschbar
```

5.2.1 Index-sequentielle Dateiverwaltung mit der Seek-Funktion

Eine der wichtigsten Funktionen haben wir bislang noch nicht behandelt: Die Seek-Routine. Sie ist die Grundlage dafür, daß man auf dem Amiga professionelle Dateiverwaltung betreiben kann, indem man den Schreib-Lesezeiger innerhalb einer Datei frei setzen kann.

Wenn Sie eine Datei geöffnet haben, wird vom DOS ein interner Pointer angelegt, der die jeweilige Schreib-Leseposition innerhalb der Datei angibt. Am Anfang des Datenzugriffs steht er auf Null, mit jedem gelesenen/geschriebenen Byte wird er um eins erhöht. Bislang sind wir davon ausgegangen, daß eine Datei immer komplett beschrieben oder ausgelesen wird, oder daß Daten angehängt werden, indem man in eine Datei hineinschreibt, die mit dem Alt-Modus geöffnet wurde. Durch die Seek-Funktion aber können wir den Pointer innerhalb der Datei völlig frei setzen, so daß wir einen Zugriff auf beliebige Dateiabschnitte haben. Hiermit ist es möglich, das Prinzip der Index-sequentiellen Dateiverwaltung anzuwenden, das ich nun kurz am Beispiel einer Adreßverwaltung erläutern möchte.

Gehen wir einmal davon aus, daß neben dem Vor- und Nachnamen noch der Wohnort und die Straße der betreffenden Person gespeichert werden sollen. Für diese einzelnen Felder innerhalb eines Datensatzes sollen die folgenden maximalen Längen festgelegt werden:

Vorname	: 20 Zeichen
Nachname	: 20 Zeichen
Postleitzahl	: 4 Zeichen
Wohnort	: 20 Zeichen
Straße	: 20 Zeichen
Gesamtlänge	: 84 Zeichen

Man kann nun hergehen und einen Datensatz nach dem anderen in die Adreßdatei hineinschreiben. Durch die Seek-Funktion hat man anschließend sofort Zugriff auf jeden einzelnen Datensatz, indem man die Position innerhalb der Datei nach der Formel

Schreib-Lese-Position = (Datensatznummer −1) * 84

berechnet, wenn am Anfang der Datei der erste Datensatz steht. Durch das Einlesen von 84 Byte hat man so den kompletten Datensatz, bestehend aus Name und Adresse, eingelesen. Wollen wir einmal annehmen, daß die Daten ab der Adresse »puffer« im Speicher stehen, so können wir einfach auf die einzelnen Datenfelder positionieren:

```
move.l #puffer,a0          ;Zeiger auf Datensatz
move.l 00(a0),d0           ;erstes Zeichen Vorname
move.l 20(a0),d1           ;erstes Zeichen Nachname
move.l 40(a0),d2           ;erstes Zeichen Postleitzahl
move.l 44(a0),d3           ;erstes Zeichen Wohnort
move.l 64(a0),d4           ;erstes Zeichen Straße
```

Das Problem besteht darin, daß man nicht weiß, welche Datensatznummer zu welcher Person gehört. Um eine bestimmte Person herauszufinden, müßte man der Reihe nach alle Datensätze einlesen und den Namen mit dem Suchkriterium vergleichen. Dieses Verfahren ist aber wegen des enormen Zeitbedarfs nicht zu empfehlen.

^Einige schlaue Köpfe, liebe Leser, haben sich daher die Methode der Index-sequentiellen Dateiverwaltung einfallen lassen. Diese zeichnet sich dadurch aus, daß neben der eigentlichen Adreßdatei noch eine weitere Datei mit dem sogenannten Index besteht. Als Index bezeichnet man ein beliebiges Feld des Datensatzes, nach dem man meistens suchen möchte, hier wäre z.B. der Nachname sinnvoll, weniger die Postleitzahl. Dieses Feld wird zusammen mit der Nummer des Datensatzes in der Indexdatei gespeichert, welche beim Programmbeginn jeweils komplett eingelesen wird. Wenn man nun einen bestimmten Namen sucht, wird nicht mehr die Adreßdatei, sondern die Indexdatei durchsucht, als Ergebnis erhält man die Nummer des zugehörigen Datensatzes in der Adreßdatei und kann so sehr schnell die weiteren Felder des Datensatzes laden:

```
Inhalt Indexdatei
Name     Datensatznummer
....     ...............
Maier    3
Müller   13
Schulze  11
```

1) Gesuchter Name: Müller
2) Gefundene Datensatznummer: 13
3) Zeigerposition innerhalb der Adreßdatei: (13-1)*84 = 1008
4) Einlesen von 84 Byte ab der Position 1008

Der Vorteil der index-sequentiellen Methode besteht darin, daß man einerseits blitzschnell ohne jeden Diskettenzugriff nach dem Indexfeld suchen kann, da sich die Indexdatei komplett im Speicher befindet, dieser aber andererseits nicht mit der Datenmenge der kompletten Datensätze belastet wird, in unserem Fall würde der Speicher mit nur (20/84)*100 = 23,8 Prozent der Gesamtdatenmenge gefüllt. Falls Ihre Adreßverwaltung z.B. 200 Kbyte lang wäre, hätten Sie noch 212 Kbyte frei für die Indexdatei und könnten eine Adreßdatei von 890,4 Kbyte verwalten, dies ist mehr als Sie überhaupt auf einer Diskette speichern können!

Problematisch wird die Sache dann, wenn Sie nach einem anderen Datenfeld suchen möchten. Hier müssen Sie dann in den sauren Apfel beißen und die gesamte Adreßdatei durchsuchen. Man kann dadurch Abhilfe schaffen, daß man weitere Indexdateien für andere Datenfelder anlegt, wobei abhängig von dem zur Verfügung stehenden Speicher auch mehrere im Speicher gehalten werden können.

Die index-sequentielle Dateiverwaltung macht sich je vorteilhafter bemerkbar, desto weniger Platz das Indexfeld im Vergleich zum gesamten Datensatz benötigt. So könnte man in unserem Beispiel mit dem Indexfeld »Postleitzahl« eine Datei verwalten, die nochmals viermal größer ist, als es mit dem Indexfeld »Nachname« möglich wäre, nämlich 3,5616 Megabyte! Falls Sie ihren Amiga sogar auf 8 Megabyte aufrüsten, könnten Sie sogar 131,04 Megabyte verwalten, was auch für ein Großunternehmen reichen sollte!

Normalerweise würde es ausreichen, wenn man den Dateipointer ausgehend vom Start der Datei setzten könnte. Die Seek-Funktion bietet aber zusätzlich die Möglichkeit, ihn ausgehend von der momentanen Position oder dem Dateiende zu setzen. Dafür muß ein Modus-Flag übergeben werden:

Modus-Flag	Bedeutung
-1	Positionierung ausgehend vom Dateistart
0	Positionierung ausgehend von der momentanen Position
+1	Positionierung ausgehend vom Dateiende

```
move.l filepointer,d1         ;Filepointer (von Open-Funktion)
move.l #position,d2           ;Position des Schreib-Lese-Zeigers
move.l #modus,d3              ;Modus-Flag
CALLDOS Seek                  ;Zeiger positionieren
```

Als Ergebnis erhält man die alte Position des Zeigers zurück.

5.3 Laden/Speichern eines IFF-Bildes mit komfortabler Fileauswahl

In professionellen Programmen ist es inzwischen zum Standard geworden, den verwöhnten Anwender nicht mehr dazu zu zwingen, Filenamen selbständig einzugeben, wie bei unserem Chart-Programm, sondern ihm eine Auswahl anzubieten, aus der er per Mausklick das gewünschte Programm auswählen kann.

Jenes Verfahren wollen wir in unserem DOS-Beispielprogramm nachvollziehen, wobei der Aufbau wieder so modular gehalten ist, daß Sie die Routine ohne weiteres in Ihre eigenen Programme übernehmen können. Allein zum Selbstzweck wäre eine solche Routine aber nicht sehr sinnvoll, daher kann das Programm nicht nur eine Datei auswählen, sondern auch einlesen und schreiben. Weil wir für das IFF-Bildformat noch kein Beispielprogramm hatten, konnten wir zwei Fliegen mit einer Klappe schlagen.

Nach dem Start des Programms »IFF-Fileauswahl« vom CLI oder von der Workbench erscheint das Startwindow mit zwei Gadgets, die zur Auswahl des Modus Lesen/Schreiben dient. Da wir noch kein Bild geladen haben, klicken wir das Gadget »Laden« an. Nun wird ein weiteres Window geöffnet, in dem die eigentliche Fileauswahl vollzogen wird (Bild 5.2).

Zunächst fallen fünf weitere Gadgets ins Auge: Zwei String-Gadgets für die Eingabe des Pfadnamens (beim Programmstart auf DF0: gesetzt) und des Dateinamens, ein noch zu erklärendes Proportionalgadget sowie zwei Boolean-Gadgets mit denen man den Ladevorgang starten kann (»Ausführen«) oder einen Abbruch erzielen kann. Das Floppylaufwerk beginnt zu laufen, worauf nach einiger Zeit die gefundenen Files und Directorys (Suffix (DIR)) angezeigt werden. Am Proportionalgadget können Sie ablesen, ob mehr Files gefunden wurden, als der Bildschirm fassen kann (10 Files). Dann wird nämlich das Feld nicht komplett ausgefüllt, Sie können durch das Anklicken den Autoknob verschieben, worauf die weiteren gefundenen Files angezeigt werden. Der Gag bei der ganzen Sache besteht nun darin, daß einfach durch Anklicken des gewünschten Filenamens dieser in dem Feld »Dateiname« erscheint und das File dann durch einen weiteren Klick auf das »Ausführen«-Gadget geladen werden kann. Die IFF-Bilder auf der mitgelieferten Diskette erkennen Sie übrigens an dem Suffix ».IFF«. Was aber kann man tun, wenn das gewünschte File nicht auf dem Bildschirm erscheint? Keine Panik, wahrscheinlich ist es nur in einem Directory versteckt. Klicken Sie doch einfach ein Directory-Gadget an, worauf das Laufwerk erneut zu laufen beginnt und Ihnen alle Einträge des Directorys anzeigt werden. Natürlich können Sie auch das Laufwerk wechseln, indem Sie die »DF0«: in eine »DF1:« verwandeln. Zur Not können Sie aber auch ganz herkömmlich den Filenamen per Hand in das Dateinamen-Feld eintragen und hoffen, daß es geladen werden kann (genauso kann natürlich auch der Pfadname eingegeben werden).

Nach dem Anklicken des »Ausführen«-Gadgtes oder dem Druck der Return-Taste wird nun versucht, das IFF-Bild zu laden. Falls die Datei gefunden werden konnte, wird ein Screen geöffnet, dessen Ausmaße sich nach den Informationen der IFF-Datei richten. Mit

Sicherheit werden Sie also Ihr Bild, anders als bei vielen anderen Programmen, im richtigen Format sehen können. Haben Sie genug gesehen, eventuell mit einem anderen Task eine Hardcopy angefertigt, können Sie den Screen nach hinten klicken und sehen wieder das Hauptmenü vor sich. Entweder speichern Sie das Bild auf die gleiche Art und Weise ab oder beenden das Programm durch das Close-Gadget.

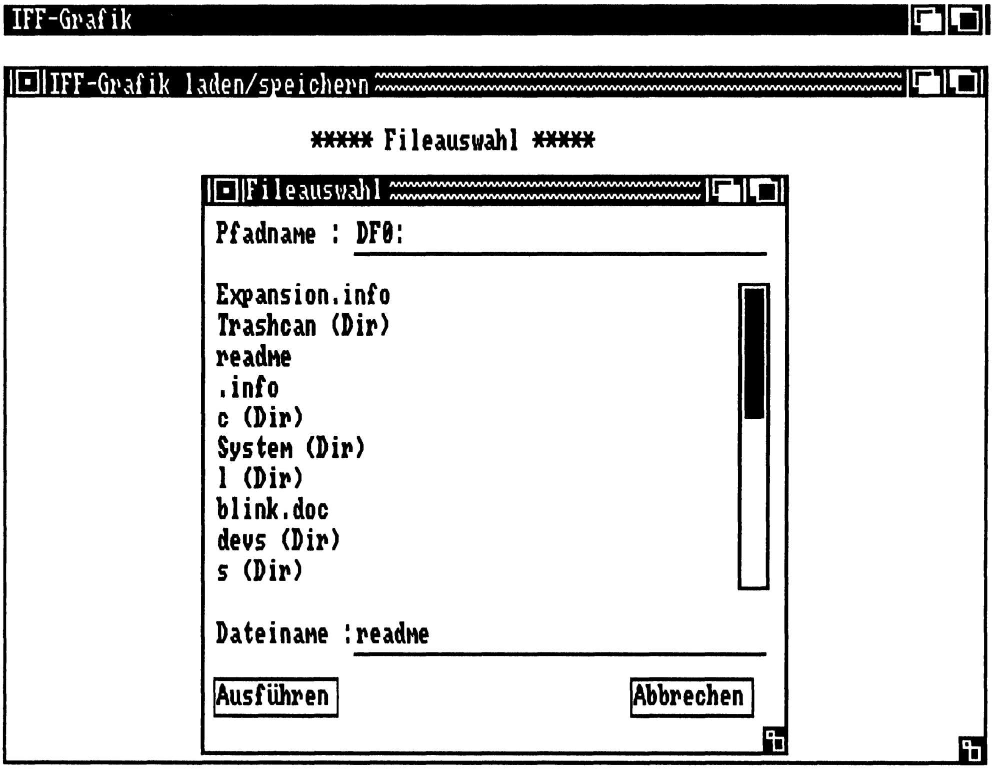


Bild 5.2: Komfortable Fileauswahl durch GADGETS

Sie sehen, liebe Leser, auch aus diesem Programm können Sie viele Elemente in Ihren eigenen verwenden. Bitte versuchen Sie aber nicht, De-Luxe-Paint-Bilder zu laden, da sie vom IFF-Standard abweichen. Zur Konvertierung in dieses Format kann man das Programm »Graficraft« verwenden, welches einerseits in der Lage ist, Deluxe-Paint-Bilder zu laden, andererseits dann aber im IFF-Format abspeichert. Auch bei dem nun folgenden Quelltext wurde aus Platzgründen auf die Ausführung der Include-Files verzichtet.

5.3.1 Include-Files, Makrodefinitionen, Öffnen der Libraries

HiSoft GenAmiga Assembler 1.21 "IFF-Fileauswahl"

```
        opt l-,d+,s-,n+
        plen 72
        llen 80


* INCLUDE-Files laden

        include df0:include/exec/exec_lib.i
        include df0:include/intuition/intuition_lib.i
        include df0:include/graphics/graphics_lib.i
        include df0:include/libraries/dos_lib.i

* MACRO-Definitionen

* Intuition-Text ausgeben

PRINTOUT        macro
                lea \1,a1
                move.l #\2,d0
                move.l #\3,d1
                move.l windowpointer,a0
                move.l 50(a0),a0
                CALLINT PrintIText
                endm

*Struktur für Ausgabe-Text

TEXTOUT         macro
                dc.b 3,0          ;Farben
                dc.b  0           ;Farbmodus
                even
                dc.w  0           ;x-Koordinate
                dc.w  0           ;y-Koordinate
                dc.l  0           ;Zeichensatz
                dc.l  \1          ;Textpointer
                dc.l  0           ;weiterer Text
                endm

* Struktur für Windows

WINDOW       macro
             dc.w      \1       ;linke Ecke
             dc.w      \2       ;obere Ecke
```

```
            dc.w    \3      ;Breite
            dc.w    \4      ;Höhe
            dc.b    0,1     ;Farben
            dc.l    \5      ;Flags für Events
            dc.l    \9      ;Window Flags
            dc.l    \6      ;Zeiger auf erstes Gadget
            dc.l    0       ;Check Mark
            dc.l    \7      ;Windowname
\8          dc.l    0       ;Screenpointer
            dc.l    0       ;Bit Map
            dc.w    \3      ;Mindest-Breite
            dc.w    \4      ;Mindest-Höhe
            dc.w    \3      ;maximale Breite
            dc.w    \4      ;maximale Höhe
            dc.w    15      ;Window-Typ
            endm

* Struktur für Screens

SCREEN          macro
           dc.w     0       ;linke Ecke
           dc.w     0       ;obere Ecke
           dc.w     \1      ;Breite
           dc.w     \2      ;Höhe
           dc.w     2       ;Tiefe (Bitplanes)
           dc.b     0,1     ;Farben
           dc.w     \3      ;Modus
           dc.w     15      ;Screentyp
           dc.l     0       ;Zeichensatz
           dc.l     \4      ;Screentitel
           dc.l     0       ;Gadgets
           dc.l     0       ;Bitmap
                endm

* Struktur für Gadgets (alle außer Proportionalgadet)

GADGET    macro
          dc.l      \1   ;nächstes Gadget
          dc.w      \2   ;obere Ecke x
          dc.w      \3   ;obere Ecke y
          dc.w     256   ;Breite
          dc.w      10   ;Höhe
          dc.w      \4   ;Flags
          dc.w       1   ;Activation-Flags
          dc.w      \5   ;Gadget-Typ
          dc.l      \6   ;Gadget-Border
          dc.l       0   ;kein Select-Gadget
```

```
         dc.l       \7    ;Gadget-Text
         dc.l        0    ;kein Exclude
         dc.l       \8    ;Special-Info
         dc.w       \9    ;Gadget-ID
         dc.l        0    ;User-Data
         endm

* Struktur für Proportionalgadget

PROPGADGET   macro
         dc.l       \1    ;nächstes Gadget
         dc.w       \2    ;obere Ecke x
         dc.w       \3    ;obere Ecke y
         dc.w       20    ;Breite
         dc.w      100    ;Höhe
         dc.w       \4    ;Flags
         dc.w        2    ;Activation-Flags
         dc.w       \5    ;Gadget-Typ
         dc.l       \6    ;Gadget-Border
         dc.l        0    ;kein Select-Gadget
         dc.l       \7    ;Gadget-Text
         dc.l        0    ;kein Exclude
         dc.l       \8    ;Special-Info
         dc.w       \9    ;Gadget-ID
         dc.l        0    ;User-Data
         endm


* Struktur für Eingaben bei Stringgadgets

STRINF           macro
                 dc.l  \1        ;Textpuffer
                 dc.l  undo      ;Undo-Puffer
                 dc.w  0         ;Cursor-Position
                 dc.w  32        ;Anzahl Zeichen (max)
                 dc.w  0         ;Ausgabeposition Cursor
                 dc.w  0         ;Position Undo-Puffer
                 dc.w  0         ;Anzahl Zeichen im Textpuffer
                 dc.w  0         ;Anzahl sichtbare Zeichen
                 dc.w  0         ;horizontaler Offset
                 dc.w  0         ;vertikaler Offset
                 dc.l  0         ;Zeiger auf Rastport
                 dc.l  0         ;Integer-Wert
                 dc.l  0         ;Tastatur-Belegung
                 endm
```

```
         * Textstruktur für Gadgets

         STRING          macro
                         dc.b  1,0          ;Farben
                         dc.b  1            ;Farbmodus
                 even
                         dc.w  \1,\2        ;Positionen
                         dc.l  0            ;Zeichensatz
                         dc.l  \3           ;Textpointer
                         dc.l  \4           ;nächster Text
                         endm

0000006E undo    dc.l  0,0,0,0              ;Undo-Puffer für Stringgadets


0000007E begin   lea intname,a1             ;Intuition-Library öffnen
00000084         clr.l d0
00000086         CALLEXEC OpenLibrary
0000008E         tst.l d0
00000090         beq fini1
00000094         move.l d0,_IntuitionBase

0000009A         lea grafname,a1            ;Grafik-Library öffnen
000000A0         clr.l d0
000000A2         CALLEXEC OpenLibrary
000000AA         tst.l d0
000000AC         beq fini2
000000B0         move.l d0,_GfxBase

000000B6         lea dosname,a1             ;DOS-Library öffnen
000000BC         clr.l d0
000000BE         CALLEXEC OpenLibrary
000000C6         tst.l d0
000000C8         beq fini3
000000CC         move.l d0,_DOSBase

000000D2         lea screen_defs,a0                 ;Screen öffnen
000000D8         CALLINT OpenScreen
000000E2         move.l d0,screenpointer            ;Pointer für
000000E8         move.l d0,point1                   ;Windows
000000EE         move.l d0,point2
```

5.3.2 Menü-Aufbau und -Auswertung

```
         * Gadgetaufbau, Abfrage
000000F4        lea startwindow,a0          ;Startwindow öffnen
000000FA        CALLINT OpenWindow
00000104        move.l d0,windowpointer
0000010A        bsr print                   ;Text ausgeben

0000010E wait   move.l windowpointer,a0
00000114        bsr message                 ;Gadget-ID holen
00000118        cmpi.l #$200,d6             ;Ende
0000011E        beq ende
00000122        cmpi.w #0,d7                ;Gadget-ID prüfen
00000126        beq load                    ;Grafik laden
0000012A        bra save                    ;Grafik speichern


         * Grafik laden

0000012E load   bsr holefile                ;Fileauswahl treffen
00000132        cmpi.b #255,d7              ;Ausführen?
00000136        bne wait                    ;Nein
0000013A        bsr gload                   ;Grafik laden
0000013E        bra wait                    ;von vorne

         * Grafik speichern

00000142 save   bsr holefile                ;Fileauswahl treffen
00000146        cmpi.b #255,d7              ;Ausführen?
0000014A        bne wait                    ;Nein
0000014E        bsr gsave                   ;Grafik speichern
00000152        bra wait
```

Bislang haben wir prinzipiell nichts Neues kennengelernt: Nachdem der Task bis zur Übermittlung einer Message schlafen gelegt wurde, wird dann auf Grund der Gadget-ID in die Lade- bzw. Speicherroutine verzweigt. Dort wird dann in das Unterprogramm »holefile« gesprungen, das die komplette Fileauswahl erledigt. Im Register D7 wird uns mitgeteilt, ob abgebrochen (0) oder die Aktion ausgeführt werden soll (255).

5.3.3 Fileauswahl treffen

HiSoft GenAmiga Assembler 1.21 "IFF-Fileauswahl"

```
         * Fileauswahl treffen
00000156 holefile lea waehlwindow,a0          ;Auswahlwindow öffnen
0000015C          CALLINT OpenWindow
00000166          move.l d0,iwindowpointer
0000016C start    clr.b  startf
00000172          lea propinfo,a5             ;Prop-Info-Struktur
00000178          clr.w  4(a5)                ;y-Position auf Start
0000017C          bsr read                    ;Files einlesen
00000180          move.b anzahl,d0
00000186          beq voll                    ;gar nichts gefunden
0000018A          cmpi.b #10,d0               ;Bei mehr als 10 Files
0000018E          bls voll
00000192          clr.l d1                    ;muß gescrollt werden
00000194          move.b anzahl,d1
0000019A          move.l #$9fff6,d0           ;Propgadget-Höhe:
000001A0          divu d1,d0                  ;(10*$FFFF)/Anzahl
000001A2          bra weiter
000001A6 voll     move.w #$FFFF,d0            ;Prop-Gadget gefüllt (<10 Files)
000001AA weiter   move.w d0,8(a5)             ;Wert in Prop-Gadget Struktur
000001AE start1   bsr anzeige                 ;Gadgets anzeigen
000001B2 start2   move.l iwindowpointer,a0
000001B8          bsr message                 ;Gadget-ID holen
000001BC          cmpi.w #10,d7               ;Name angeklickt?
000001C0          bls noname                  ;nein

         * Eines der 10 Filenamen-Gadgets an geklickt

000001C4 name     subi.w #100,d7               ;ID auf 0-9 reduzieren
000001C8          add.b startf,d7              ;plus Bildschirmstartfile
000001CE          mulu #32,d7                  ;mal 32 (Länge Textpuffer)
000001D2          addi.l #fpuffer,d7           ;plus Textpuffer-Start
000001D8          move.l d7,a0
000001DA          move.b 31(a0),d0             ;Directory?
000001DE          bmi pfad                     ;Ja, Pfadnamen ändern
000001E2          lea Dateiname,a1             ;Zeiger auf Stringpuffer
000001E8          move.w #31,d0
000001EC copy4    move.b #32,(a1)+
000001F0          subq.w #1,d0                 ;Puffer mit Leerzeichen
000001F2          bne copy4                    ;füllen
000001F6          clr.b (a1)+                  ;Gadgets löschen
000001F8          lea Dateiname,a1             ;Start Textpuffer Dateiname
000001FE Loop1    move.b (a0),(a1)+            ;Daten kopieren
00000200          cmpi.b #0,(a0)+              ;bei Nullbyte Ende des Strings
```

```
00000204         bne Loop1
00000208         bra start1                 ;Gadgets neu ausgeben


         * Pfadname, Dateiname, Abbruch oder Ausführen angeklickt

0000020C noname  mulu #4,d7                 ;mal 4 wegen Langwörtern
00000210         addi.l #tab,d7             ;plus Tabellenstart
00000216         move.l d7,a0
00000218         move.l (a0),a0             ;Adresse der Routine holen
0000021A         jmp (a0)                   ;Routine anspringen

0000021C tab     dc.l pfad1,ausführen,abbrechen,propgadget,file

         * Proportionalgadget angeklickt

00000230 propgadget cmpi.b #9,anzahl        ;Anzahl Files kleiner als 10
00000238            bls start2              ;Dann kein neuer Aufbau
0000023C            lea propinfo,a5         ;Prop-Info-Struktur
00000242            move.w 4(a5),d0         ;y-Position holen
00000246            clr.w d1                ;(Anzahl Files-10)
00000248            move.b anzahl,d1        ;*Prop-Position/$FFFF
0000024E            subi.b #10,d1
00000252            mulu d1,d0
00000254            divu #$FFFF,d0          ;= Startfile
00000258            move.b d0,startf
0000025E            bra start1              ;Gadgets neu ausgeben

         * Dateiname angeklickt

00000262 file       bra ausführen           ;File einlesen

         * Abbruch angeklickt

00000266 abbrechen  clr.b   d7              ;Flag für Abbruch
00000268            bra fertig

         * Ausführen angeklickt

0000026C ausführen  move.b #255,d7          ;Flag für Ausführen
00000270            bra fertig

         * Pfadname angeklickt

00000274 pfad    lea Pfadname,a1            ;Start Textpuffer Pfadname
0000027A         move.b #-1,d0
0000027E Loopa   addi.b #1,d0
```

```
00000282          cmpi.b #0,(a1)+          ;feststellen
00000286          bne Loopa
0000028A          move.b -(a1),d1
0000028C          cmpi.b #4,d0             ;Bisher nur Drivename?
00000290          beq Loopb                ;ja
00000294          move.b #"/",(a1)+        ;sonst Trennstrich anhängen
00000298 Loopb    move.b (a0),(a1)+        ;Daten kopieren
0000029A          cmpi.b #0,(a0)+          ;bei Nullbyte Ende des Strings
0000029E          bne Loopb
000002A2          move.w #7,d0
000002A6 Loopc    clr.b  -(a1)             ;"(Dir)" löschen
000002A8          subq.w #1,d0
000002AA          bne Loopc

000002AE pfad1    bra start                ;neue Files einlesen


         * Files zur Auswahl einlesen

000002B2 read      lea fpuffer,a1          ;Zeiger auf Puffer
000002B8           move.w #3200,d0         ;3200 Byte löschen
000002BC copy6     clr.b (a1)+
000002BE           subq.w #1,d0            ;Puffer mit Leerzeichen
000002C0           bne copy6
000002C4           clr.b anzahl            ;Initialisierung
000002CA           move.l #Pfadname,d1     ;Diskette untersuchen
000002D0           move.l #-2,d2
000002D6           CALLDOS Lock            ;(Schlüssel suchen)
000002E0           tst.l d0
000002E2           beq endofread           ;kein Schlüssel gefunden
000002E6           move.l d0,locksave      ;merken
000002EC           move.l d0,d1
000002EE           move.l #fileinfo,d2     ;Zeiger auf File-Info-Block
000002F4           CALLDOS Examine         ;Diskette weiter untersuchen
000002FE           tst.l d0
00000300           beq endofread           ;nichts gefunden
00000304           move.l #fpuffer,d5      ;Zeiger auf Puffer für Filenamen
0000030A readloop  move.l locksave,d1      ;Fileeinträge einlesen
00000310           move.l #fileinfo,d2     ;in Fileinfo-Block
00000316           CALLDOS ExNext
00000320           tst.l d0
00000322           beq endofread           ;keine Einträge mehr gefunden
00000326           lea fileinfo+8,a4       ;Zeiger auf File-Info-Block
0000032C           addi.b #1,anzahl        ;Anzahl der Files erhöhen
00000334           move.l d5,a3
00000336 trans     move.b (a4),(a3)+       ;Puffer kopieren
00000338           cmpi.b #0,(a4)+         ;letztes Zeichen gefunden,
```

```
0000033C            bne trans               ;wenn Null
00000340 endofname move.l fileinfo+4,d0     ;File oder Directory?
00000346            bmi datei               ;Directory gefunden, dann
0000034A            move.b -(a3),d0
0000034C            clr.l d0                ;"Dir" zu Namen hinzufügen
0000034E            lea text,a0
00000354 rloop      move.b (a0)+,(a3)+      ;Text anfügen
00000356            addi.b #1,d0
0000035A            cmpi.b #6,d0
0000035E            bne rloop
00000362            clr.b (a3)              ;Filenamen abschließen
00000364            move.l d5,a3
00000366            move.b #$FF,31(a3)      ;Kennzeichen für Directory
0000036C datei      addi.l #32,d5           ;Zeiger in Puffer erhöhen
00000372            bra readloop
00000376 endofread rts

         * Gadgets in Window anzeigen

00000378 anzeige    lea string1000,a1       ;Zeiger auf ersten Stringpuffer
0000037E            move.w #10,d3           ;10*34 Byte löschen
00000382 copy3      move.w #33,d0
00000386 copy2      move.b #32,(a1)+
0000038A            subq.w #1,d0            ;Puffer mit Leerzeichen
0000038C            bne copy2
00000390            clr.b (a1)+             ;alle Gadgets löschen
00000392            subq.w #1,d3            ;löschen
00000394            bne copy3
00000398            move.b anzahl,d0
0000039E            beq full                ;keine Files vorhanden
000003A2            clr.l d1
000003A4            move.b #1,d2            ;Zähler für angezeigte Files
000003A8            move.b startf,d1
000003AE            mulu #32,d1             ;Position im Filenamenpuffer
000003B2            addi.l #fpuffer,d1      ;errechnen
000003B8            move.l d1,d4            ;Pointer auf Stringpuffer
000003BA            move.b startf,d1
000003C0            move.l #string1000,d5   ;Start von Textpuffer 1.Gadget
000003C6 copy1      move.l d5,a2
000003C8            move.l d4,a0
000003CA copy       move.b (a0),(a2)+       ;Daten kopieren
000003CC            cmpi.b #0,(a0)+         ;bis Nullbyte gefunden
000003D0            bne copy
000003D4            move.b #32,-(a2)        ;Nullbyte löschen
000003D8            move.l d5,a2
000003DA            clr.b 31(a2)            ;Abschluß mit Nullbyte
000003DE            addi.b #1,d2            ;Zahl der angezeigten Files
```

```
000003E2          cmpi.b #10,d2              ;größer als 10, dann
000003E6          bhi full                   ;Bildschirm voll
000003EA          addi.b #1,d1               ;alle Files augegeben,
000003EE          cmp.b anzahl,d1            ;dann ebenfalls Ausgabestop
000003F4          bge full
000003F8          addi.l #34,d5
000003FE          addi.l #32,d4              ;Pointer erhöhen
00000404          bra copy1                  ;nächstes File
00000408 full     move.l iwindowpointer,a1 ;Zeiger auf Windowpointer
0000040E          move.l $3E(a1),a0          ;Zeiger auf erstes zu
00000412          move.l #0,a2               ;refreshendes Gadget
00000418          CALLINT RefreshGadgets
00000422          rts


00000424 fertig  move.l iwindowpointer,a0   ;Window schließen
0000042A         CALLINT CloseWindow
00000434         rts

00000436          cnop 0,4
00000438 pointer  dc.l 0                     ;Pointer auf ausgewähltes File
0000043C fileinfo ds.b 260                   ;Speicher für File-Info-Block
00000540 fpuffer  ds.l 800                   ;Puffer für max. 100 Filenamen
000011C0 locksave dc.l 0                     ;Speicher für Diskettenschlüssel
000011C4 startf   dc.b 0                     ;Erstes, auszugebendes File
                  even
000011C6 anzahl   dc.b 0                     ;Zahl der gefundenen Files
                  even
000011C8 text     dc.b " (Dir)"
                  even
000011CE altpfad  dc.l 0                     ;Zeiger auf alte Lock-Struktur
000011D2 rpuff    ds.l 28                    ;Zeiger auf Lese-Schreibpuffer
00001242 heigth   dc.w 0                     ;Screen-Höhe
00001244 depth    dc.w 0                     ;Screentiefe
00001246 widthd8  dc.w 0                     ;Screenbreite/8
                  cnop 0,4
00001248 filepointer dc.l 0
```

Zunächst zum Einlesen der Files in den Speicher: Nachdem das Window geöffnet wurde, wird mit Hilfe der Lock-Funktion der Schlüssel zum gewünschten Pfadnamen gesucht. Falls er gefunden wurde, wird zunächst der erste passende Eintrag durch die Examine-Funktion geholt. Da es sich dabei aber um Daten der Diskette und nicht eines Files handelt, werden sie ignoriert. Der erste sinnvolle Datenblock wird daraufhin mit der ExNext-Funktion geholt. Weil uns nur der Filename interessiert, wird dieser ab der Adresse 8 im FileInfo-Block in einen von uns für die Filenamen reservierten Puffer hineinkopiert. Dabei

wird byteweise solange kopiert, bis ein Nullbyte gefunden wird, welches das Ende des Filenamens angibt. Abschließend wird noch an Hand des Filetyp-Langwortes, das im FileInfo-Block ab der Adresse 4 steht, prüfen, ob es sich bei der gefundenen Datei um ein File oder ein Directory handelt. Bei einem Directory nämlich wollen wir noch den Text »(Dir)« an den Namen anhängen, um dem Anwender zu verdeutlichen, daß er diese Datei nicht laden, wohl aber als Pfadnamen verwenden kann. Diese Prozedur wiederholt sich solange, bis die ExNext-Funktion den Wert Null zurückgibt und damit anzeigt, daß kein weiterer Eintrag gefunden wurde. Der Puffer für die Filenamen enthält 3200 Byte, so daß 100 Einträge behandelt werden können. Wenn Sie meinen, daß Platz für 200 Files notwendig ist, ändern Sie einfach den DEVPAC-Befehl »ds.b 3200« in »ds.b 6400« ab.

Damit hat das Unterprogramm »read« schon seine Schuldigkeit getan. Als nächstes wird das Proprotionalgadget initialisiert, dessen Größe ins Verhältnis zu der Anzahl der gefundenen Files gesetzt werden muß. Wie Sie wissen, bewegt sich die Länge des Autoknobs zwischen 0 und $FFFF. Da wir immer gleichzeitig zehn Filenamen auf den Bildschirm bringen können, gilt für die Länge des Autoknobs.

```
Länge = $FFFF * (10/Zahl Files) = 10 * $FFFF/Zahl Files = $9FFF6/Zahl Files
```

Dieses Verfahren funktioniert aber nur solange, bis die Zahl der gefundenen Files größer oder gleich 10 ist, sonst würde ein illegaler Wert, der größer als $FFFF ist, herauskommen. Wir prüfen daher die Anzahl der Files auf 10, und legen die Länge des Autoknos zu $FFFF fest, wenn weniger als 10 Einträge gefunden wurden. Anschließend wird dieses Wort in die Prop-Info-Strukur eingetragen.

Im Unterprogramm »anzeige« werden jetzt die ersten zehn Filenamen ausgegeben. Sie haben wahrscheinlich schon vermutet, daß sie als Boolean-Gadgets ausgeführt sind, damit man auf das Anklicken mit der Maus reagieren kann. Die andere Möglichkeit wäre gewesen, die Dateinamen einfach auf den Bildschirm zu schreiben und anschließend abhängig von der Mausposition auf ein bestimmtes File rückzuschließen. Hierfür wäre aber deutlich mehr Programmieraufwand nötig gewesen als bei der von mir gewählten Methode, die Filenamen Gadgets zuzuordnen. Zunächst muß der Textpuffer der Gadgets, der bei »string1000« beginnt, gelöscht werden. Den Grund hierfür nenne ich Ihnen unten. Dann werden abhängig von der Position des Proportionalgadgets zehn Dateinamen aus dem Filenamen-Puffer in den Gadget-Puffer kopiert. Die Proportionalgadgetstellung findet sich indirekt in dem Speicher »startf« wieder, in dem sich die Nummer des Files befindet, das als erstes ausgegeben wird. Wie man diese Nummer aus der Gadgetstellung berechnen kann, sehen wir unten.

Der Vorgang wird solange wiederholt, bis entweder zehn Filenamen kopiert wurden und der Bildschirm gefüllt ist, oder das letzte überhaupt existierende File behandelt wurde, es ist ja nicht immer gesagt, daß überhaupt zehn Dateien gefunden wurden, in diesem Fall wird der Bildschirm eben nur teilweise belegt.

Als letztes wird ein Refreshing der Gadgets durchgeführt. Dies bedeutet nichts anderes, als das alle Daten wie Text, Größe, Position etc. neu gesetzt und eine neue Bildschirmausgabe

auf Grund der geänderten Daten vorgenommen werden. Uns interessiert natürlich nur der geänderte Gadget-Text. Das Problem beim Refresh besteht darin, daß diese Funktion im Prinzip fehlerhaft arbeitet. So wird zwar der neue Gadget-Text ausgegeben, ohne sich aber um den vorhergehenden Text zu kümmern, dieser wird keineswegs gelöscht. Wenn Ihr neuer Text daher kürzer ist als der alte, wird er nur bis zu der Position überschrieben, die durch die Länge des neuen Textes vorgegeben ist. Der Rest des alten Textes bleibt als unerwünschte Ruine bestehen, wie folgendes Beispiel zeigt:

Inhalt Gadget-Text-Puffer vor Refresh:	`"Commodore Amiga,                 0"`
Bildschirminhalt vor Refresh:	`Commodore Amiga`
Inhalt Gadget-Text-Puffer nach Refresh:	`"Amiga,                           0"`
Bildschirminhalt nach Refresh:	`Amigadore Amiga`

Deshalb war es erforderlich, am Anfang der Ausgaberoutine den Gadget-Text-Puffer mit Leerzeichen komplett zu löschen und erst ganz am Ende das Nullbyte als Kennzeichen des Textendes zu setzen. Dadurch werden alle Leerzeichen als Bestandteil des Textes aufgefaßt und ausgegeben, wodurch der alte Text wirklich gelöscht wird:

Inhalt Gadget-Text-Puffer vor Refresh:	`"Commodore Amiga,                 0"`
Bildschirminhalt vor Refresh:	`Commodore Amiga`
Inhalt Gadget-Text-Puffer nach Refresh:	`"Amiga,                           0"`
Bildschirminhalt nach Refresh:	`Amiga`

Die Gadget-Refresh-Funktion benötigt neben dem Zeiger auf die Window-Struktur noch einen weiteren auf die zu refreshende Gadget-Struktur, den man aus der Window-Strukur ab der Adresse $3E auslesen kann.

Der folgende Programmteil beschäftigt sich mit der Auswertung des Message-Portes und zeigt, wie man durch geschickte Wahl der Gadget-IDs viel Programmieraufwand sparen kann. Zunächst unterscheiden wir zwischen den zehn Gadgets, durch die die Filenamen repräsentiert werden, und den übrigen fünf Gadgets.

Die Reaktion auf das Anklicken eines Filenamen-Gadgtes ist prinzipiell immer gleich, da der Filename in den Textpuffer für das zu ladende File kopiert werden muß. Die Startadresse innerhalb des Textpuffers der Gadgtes berechnet sich bei einer Pufferlänge von 32 Zeichen für jedes Gadget zu

Adresse = (gewählte Filenummer (1-10) -1)*32

Indem wir unseren Gadgets fortlaufende IDs gegeben haben (100-109), können wir nun einfach durch die Subtraktion des Offsets 100 auf die Filenummer und somit auch auf die Adresse im Textpuffer schließen, ab der kopiert werden muß. Nun kann der Inhalt des betreffenden Textpuffers in den des Dateinamen-Gadgets kopiert werden, wobei dieser aus den oben genannten Gründen zunächst wieder mit Leerzeichen gelöscht wird.

Die Auswertung der übrigen fünf Gadgets verläuft ähnlich wie bei den Menüpunkten in den vorigen Beispielprogrammen. Aus der Gadget-ID wird ein Offset berechnet, mit dessen Hilfe man die Startadresse der entsprechenden Routine aus einer Tabelle auslesen kann. Im Gegensatz zu den Menüs haben wir es hier nur mit einer Tabelle zu tun, weil die Gadgets nur eindimensional und nicht wie die Menüs zweidimensional (Menü, Menüpunkt) aufgebaut sind. Das hier verwendete Verfahren bietet sich immer dann an, verwendet zu werden, wenn man mehrere Gadgets auswerten muß, so ab vier Stück lohnt sich die Sache.

Durch das Proportionalgadget wird das erste, auf dem Bildschirm auszugebende File angewählt. Um an dessen Nummer zu kommen, kann man folgende Verhältnisgleichung benutzen:

```
starf = ((Anzahl Files - 10) * (Gadgetposition)/$FFFF
```

Nehmen wir einmal an, durch die ExNext-Funktion wurden 30 Files gefunden. Nachdem die ersten zehn Files angezeigt wurden, möchten Sie nun die nächsten 10 anzeigen. In dem Proportionalgadget-Feld ist genau ein Drittel durch den Autoknob belegt, wie die obige Formel zeigt:

Länge = $FFFF * (10/30) = 10 * $FFFF/30 = $9FFF6/30 = $5555

Leider muß ich an dieser Stelle einen weiteren Irrglauben einiger meiner Autorenkollegen korrigieren: Es ist keineswegs so, daß die Länge des Autoknobs auch der Schrittweite entspricht, was auch völlig falsch wäre, wie man sich leicht überlegen kann. Vielmehr wird die Schrittweite von Intuition nach der Formel

Schritt = $FFFF/(($FFFF/Länge) -1)

berechnet, in unserem Fall würde sich somit der Wert $7FFF ergeben. So können wir leicht das Startfile für unsere Bildschirmausgabe berechnen:

startf = ((30-10) * ($7FFF)/$FFFF = 10

Dies ist natürlich richtig, da wir zunächst die Files 0 bis 9 ausgegeben hatten und demzufolge jetzt die Files 10 bis 19 betrachten möchten. Wenn das Dateinamen-Gadget angewählt wurde, brauchen wir überhaupt nichts zu unternehmen, da der Anwender in diesem Fall selbständig den gewünschten Namen einträgt. Der Lade- bzw. Speichervorgang wird aber erst nach dem Anklicken des »Ausführen«-Gadgets begonnen. Damit sind wir auch schon bei den letzten beiden Gadgets, bei denen wir einfach nur ein Flag für das Hauptprogramm setzten, je nachdem, ob ein Abbruch erzwungen oder die Ausführung begonnen werden soll.

5.3.4 Grafik laden

HiSoft GenAmiga Assembler 1.21 "IFF-Fileauswahl"

```
         * Grafik laden

0000124C gload     move.l #-2,d2           ;Modus: Lesen
00001252           bsr newdir              ;neues Directory setzen
00001256           beq loadend             ;hat nicht geklappt, Ende
0000125A           move.l #1005,d2         ;Modus: Lesen
00001260           bsr open                ;Datei öffnen
00001264           beq loadend1            ;Fehler, altes Directory
00001268           bsr clscreen            ;IFF-Screen schließen
0000126C           move.l #12,d3
00001272           bsr lesen               ;12 Byte (Header) überlesen
00001276 reloop    moveq.l #4,d3           ;4 Byte lesen
00001278           bsr lesen
0000127C           beq loadend2            ;keine Daten vorhanden
00001280           cmpi.l #-1,d0           ;Fehler?
00001286           beq loadend2            ;ja, Ende
0000128A           cmpi.l #"BMHD",rpuff    ;Bit-Map-Header gefunden?
00001294           bne r1                  ;Nein
00001298           move.l #24,d3           ;24 Byte einlesen
0000129E           bsr lesen
000012A2           lea rpuff,a1            ;Zeiger auf Datenpuffer
000012A8           lea bild,a0             ;Zeiger auf NewScreen-Struktur
000012AE           move.w 4(a1),4(a0)      ;Breite des Screens eintragen
000012B4           move.w 6(a1),6(a0)      ;Höhe des Screens eintragen
000012BA           addi.w #10,6(a0)        ;+10 wegen Titelleiste
000012C0           move.b 12(a1),9(a0)     ;Lowbyte der Tiefe eintragen
000012C6           move.w 8(a0),depth      ;Tiefe im Wortformat merken
000012CE           clr.w 12(a0)            ;Grafikmodus normal (320*256)
000012D2           move.w 4(a0),d0         ;Breite laden
000012D6           cmpi.w #320,d0          ;Größer als 320?
000012DA           bls nohires             ;nein, kein HiRes-Modus
000012DE           ori.w #$8000,12(a0)     ;HiRes-Flag einblenden
000012E4 nohires   lsr.w #3,d0             ;Breite geteilt durch 8 merken
000012E6           move.w d0,widthd8
000012EC           move.w 6(a0),d0         ;Höhe laden
000012F0           cmpi.w #256,d0          ;Größer als 256?
000012F4           bls nointer             ;nein, kein Interlace-Modus
000012F8           ori.w #4,12(a0)         ;Interlace-Flag einblenden
000012FE nointer   move.w d0,heigth        ;Höhe merken
00001304           CALLINT OpenScreen      ;Screen öffnen
0000130E           move.l d0,bildpointer   ;Screenpointer sichern
00001314           beq loadend2            ;Fehler beim Öffnen
00001318           bra reloop              ;Weitere Daten lesen
```

```
0000131C r1       cmpi.l #"CMAP",rpuff      ;Color-Map gefunden?
00001326          bne r2                    ;Nein
0000132A          moveq.l #4,d3             ;Anzahl Farben einlesen
0000132C          bsr lesen
00001330          move.l rpuff,d7           ;durch 3 dividieren
00001336          divu #3,d7
0000133A          clr.l d6                  ;Zähler für Farbregisternummer
0000133C          lea rpuff,a5
00001342          move.l bildpointer,d0     ;Zeiger auf Screen-Struktur
00001348          addi.l #$2C,d0            ;Offset für View-Port
0000134E          move.l d0,a4              ;Zeiger merken
00001350 farbloop moveq.l #3,d3             ;für jede Farbe Rot-, Grün-
00001352          bsr lesen                 ;und Blauanteil einlesen
00001356          move.l a4,a0              ;Zeiger auf View-Port
00001358          move.l d6,d0              ;Farbregisternummer
0000135A          clr.l d1
0000135C          move.b (a5),d1            ;Rotanteil
0000135E          lsr.b #4,d1               ;/16
00001360          clr.l d2
00001362          move.b 1(a5),d2           ;Grünanteil
00001366          lsr.b #4,d2               ;/16
00001368          clr.l d3
0000136A          move.b 2(a5),d3           ;Blauanteil
0000136E          lsr.b #4,d3               ;/16
00001370          CALLGRAF SetRGB4          ;Farbe setzen
0000137A          addq.l #1,d6              ;Register erhöhen
0000137C          cmp.w d6,d7               ;Nächstes Register lesen
0000137E          bhi farbloop
00001382          bra reloop                ;weitere Daten lesen

00001386 r2       cmpi.l #"BODY",rpuff      ;Grafikdaten gefunden?
00001390          bne r3                    ;nein!
00001394          moveq.l #4,d3             ;4 Byte überlesen
00001396          bsr lesen                 ;(Grafikgröße in Bytes)
0000139A          move.l bildpointer,a0     ;Screenpointer holen
000013A0          move.l $58(a0),a5         ;Zeiger auf Bitmap aus Rastport
000013A4          clr.l d5                  ;Zähler für Höhe
000013A6 rl2      clr.l d6                  ;Zähler für Tiefe
000013A8          moveq.l #8,d4             ;Zeiger auf erste Bitplane
000013AA rl1      move.l filepointer,d1     ;Zeiger auf Filepointer
000013B0          move.l d5,d2              ;Zeile
000013B2          addi.l #10,d2             ;ab Zeile 10 wg. Titelleiste
000013B8          mulu widthd8,d2           ;* Breite/8 = Offset
000013BE          add.l (a5,d4.l),d2        ;+ Startadresse Bitplane
000013C2          clr.l d3
000013C4          move.w widthd8,d3         ;Breite/8 Byte pro Zeile
000013CA          CALLDOS Read              ;Daten einlesen
```

```
000013D4            addq.l #4,d4             ;Zeiger auf nächste Bitplane
000013D6            addq.w #1,d6             ;Tiefe erhöhen
000013D8            cmp.w depth,d6
000013DE            blt rl1                  ;nächste Bitplane
000013E2            addq.w #1,d5             ;Höhe erhöhen
000013E4            cmp.w heigth,d5
000013EA            blt rl2                  ;nächste Zeile lesen
000013EE            bra loadend2             ;Fertig

000013F2 r3         cmpi.l #"CRNG",rpuff     ;Color-Range-Daten?
000013FC            bne r4                   ;nein
00001400            move.l #12,d3            ;ja, 12 Byte überlesen
00001406            bsr lesen
0000140A            bra reloop               ;weitere Daten lesen

0000140E r4         cmpi.l #"DPPV",rpuff     ;Grafikcraft-Daten?
00001418            bne r5                   ;nein
0000141C            move.l #108,d3           ;ja, 108 Byte überlesen
00001422            bsr lesen
00001426            bra reloop               ;weitere Daten lesen

0000142A r5         cmpi.l #"CAMG",rpuff     ;View-Port-Daten?
00001434            bne r6                   ;nein
00001438            moveq.l #8,d3            ;ja, 8 Byte überlesen
0000143A            bsr lesen
0000143E            bra reloop               ;weitere Daten lesen

00001442 r6         cmpi.l #"CCRT",rpuff     ;Daten zur Farbanimation?
0000144C            bne loadend              ;nein, fertig
00001450            move.l #18,d3            ;ja, 18 Byte überlesen
00001456            bsr lesen
0000145A            bra reloop               ;weitere Daten lesen

0000145E loadend2   bsr close                ;Datei schließen
00001462 loadend1   bsr altdir               ;altes Directory wiederherstellen
00001466 loadend    rts
```

Zunächst wird die Datei geöffnet, nachem das aktuelle Directory mit der CurrentDir-Funktion auf den gewünschten Pfadnamen gesetzt wurde. Als nächstes wird ein eventuell schon bestehender Screen für das IFF-Bild geschlossen. Nachdem die Bild-Header-Daten überlesen wurden, wird ein Langwort eingelesen, das den nun folgenden Block charakterisiert. Außer dem BitMap-Header (»BMHD«), der Color-Map (»CMAP«) und den Grafikdaten (»BODY«) werden wir alle gefundenen Blöcke ignorieren. Der Bitmap-Header enthält die Breite, Höhe und Tiefe des Screens, die entsprechend in die NewScreen-Struktur eingetragen werden müssen, bevor der Screen geöffnet werden kann. In Abhängigkeit von

der Breite und der Höhe muß eventuell noch der HiRes- oder/und der Interlace-Modus gesetzt werden. Etwas unsinnig erscheint auf den ersten Blick, daß die gefundene Höhenangabe um 10 Pixel erhöht wird. Der Grund ist aber plausibel: Eine Grafik-Bitmap beginnt immer in der oberen linken Ecke des Screens. Dort liegt aber auch dessen Titelleiste mit dem Namen und den Systemgadget. Wenn wir diese nicht mit den Grafikdaten überschreiben wollen, dürfen wir erst in der zehnten Zeile mit dem Laden beginnen. Verheerend wäre es, in der zehnten Zeile zu beginnen, ohne die Höhe des Screens erhöht zu haben. Dann nämlich würde die Bitmap-Größe nur für die Grafikhöhe 10 ausgelegt, die letzten zehn Zeilen würden somit in undefinierte Speicherbereiche hineingeschrieben. Wenn der Guru nicht auftauchen sollte, würde ich Ihnen einen abendlichen Casinobesuch oder das Ausfüllen eines Lottoscheines empfehlen, denn soviel Glück kann man normalerweise gar nicht haben.

Die Color-Map enthält, wie im Kapitel 4 beschrieben, die Anzahl der Farben mal 3 und die einzelnen Farbanteile von den Grundfarben Rot, Grün und Blau. Diese können mit der SetRGB4-Funktion in das entsprechende Register eingetragen werden, nachdem sie aus dem Datenpuffer ausgelesen wurden. Schließlich zu den Grafikdaten selbst. Wie Sie schon wissen, liebe Leser, sind diese zeilenweise abgespeichert worden. Daher wird zunächst in einer inneren Schleife die jeweils erste Zeile aller vorkommenden Bitplanes geladen. Die Ladeadresse kann man nach der Formel

Startadresse = Startadresse Bitplane + (Zeilennummer+10)*(Breite/8)

berechnen. Dies ist leicht verständlich, wenn man bedenkt, daß jeweils 8 Grafikpunkte ein Byte benötigen und wir wegen der Titelleiste erst in der zehnten Zeile beginnen dürfen. Die Anzahl der zu ladenden Datenbytes pro Zeile wird ebenfalls durch die Screen-Breite/8 festgelegt. Die Startadresse der Bitplanes erhält man aus der Bitmap-Struktur, wobei die Adresse der ersten Bitplane ab der Adresse 8 zu finden ist. In jedem Schleifendurchlauf wird die Adresse innerhalb der Bitmap-Struktur um 4 erhöht, so daß nacheinander die Adressen aller Bitplanes geholt werden können. In einer äußeren Schleife schließlich wird die Zeilennummer solange um eins erhöht, bis alle Zeilen gelesen wurden. Alle weiteren gefundenen Blöcke werden wie gesagt ignoriert. Dafür wird die entsprechende Anzahl von Datenbytes gelesen, um den Lese-Pointer auf den Titel des folgenden Grafikblocks zu setzen:

"CRNG" : 12 Byte
"DPPV" : 108 Byte
"CAMG" : 8 Byte
"CCRT" : 18 Byte

Wenn keine weiteren Daten mehr gefunden werden, wird die Datei geschlossen und das aktuelle Directory auf den alten Pfadnamen zurückgesetzt, womit das Laden der Grafik abgeschlossen ist. Sie können die Grafik solange betrachten, wie Sie möchten, eine Hardcopy anfertigen, usw. Um in unserem Programm weiterzumachen, klicken Sie einfach das Systemgadget »Screen to Back« an, wonach Sie wieder das Hauptmenü vor sich sehen.

5.3.5 Grafik speichern

HiSoft GenAmiga Assembler 1.21 "IFF-Fileauswahl"

```
         * Grafik speichern

00001468 gsave     move.l #-2,d2            ;Modus: Lesen
0000146E           bsr newdir               ;neues Directory setzen
00001472           beq saveend              ;hat nicht geklappt, Ende
00001476           move.l #1006,d2          ;Modus: Schreiben
0000147C           bsr open                 ;Datei öffnen
00001480           beq saveend1             ;Fehler, altes Directory
00001484           move.l #"FORM",rpuff     ;Kopf eintragen
0000148E           lea bild,a5              ;Zeiger auf NewScreen-Struktur
00001494           move.w $04(a5),d7        ;Breite holen
00001498           lsr.w #3,d7              ;durch 8 teilen
0000149A           move.w d7,widthd8        ;merken
000014A0           subi.w #10,$06(a5)       ;Höhe-10 wegen Titelleiste
000014A6           move.w $06(a5),heigth    ;Höhe merken
000014AE           mulu heigth,d7           ;(Breite/8)*Höhe
000014B4           move.w $08(a5),depth     ;Tiefe holen
000014BC           mulu depth,d7            ;(Breite/8)*Höhe*Tiefe=Bytes
000014C2           move.l d7,d4             ;Größe der Grafik merken
000014C4           clr.w  d0                ;(2 hoch Tiefe) = Farbzahl
000014C6           moveq.l #1,d6            ;aktuelle Farbzahl
000014C8 sl1       addq.w #1,d0             ;Zähler erhöhen
000014CA           lsl.w #1,d6              ;Farben verdoppeln
000014CC           cmp.w depth,d0           ;mit Tiefe vergleichen
000014D2           bne sl1                  ;erreicht, dann fertig
000014D6 farboff   move.l d6,d5             ;Farbzahl merken
000014D8           mulu #3,d6               ;mal 3 nehmen
000014DC           add.l d6,d7              ;zur Größe der Datei addieren
000014DE           addi.l #48,d7            ;plus 48 für sonstige Bytes
000014E4           move.l d7,rpuff+4        ;Größe der Datei in Bytes
000014EA           move.l #"ILBM",rpuff+8   ;Datenanfang
000014F4           move.l #"BMHD",rpuff+12  ;Bit-Map-Header
000014FE           move.l #20,rpuff+16      ;Wert 20
00001508           move.w $04(a5),rpuff+20  ;Breite des Screens
00001510           move.w $06(a5),rpuff+22  ;Höhe des Screens
00001518           clr.l rpuff+24           ;Wert 0
0000151E           move.b $09(a5),rpuff+28  ;Tiefe des Screens
00001526           clr.b rpuff+29           ;Wert 0
0000152C           clr.l rpuff+30           ;Wert 0
00001532           move.b #10,rpuff+34      ;Wert 10
0000153A           move.b #11,rpuff+35      ;Wert 11
00001542           move.w $04(a5),rpuff+36  ;Breite des Screens
0000154A           move.w $06(a5),rpuff+38  ;Höhe des Screens
```

```
00001552           move.l #40,d3           ;40 Byte schreiben
00001558           bsr schreiben
0000155C           beq saveend2            ;keine Daten vorhanden
00001560           cmpi.l #-1,d0           ;Fehler?
00001566           beq saveend2            ;ja, Ende

0000156A           move.l #"CMAP",rpuff    ;Color-Map
00001574           move.l d6,rpuff+4       ;Farbanzahl*3 eintragen
0000157A           clr.l d7                ;Zähler für Farbregisternummer
0000157C           lea rpuff+8,a5
00001582           move.l bildpointer,a4   ;Zeiger auf Screen-Struktur
00001588           adda.l #$2C,a4          ;Offset für View-Port
0000158E           move.l 4(a4),a4         ;Zeiger auf die Color-Map
00001592 farbloop1 move.l a4,a0            ;Zeiger auf Color-Map
00001594           move.l d7,d0            ;Farbregisternummer
00001596           CALLGRAF GetRGB4
000015A0           move.b d0,d1            ;Farbcode merken
000015A2           andi.w #$0F00,d0        ;Rot-Bits isolieren
000015A6           lsr.w #4,d0             ;Rotanteil*16
000015A8           move.b d0,(a5)+         ;eintragen
000015AA           move.b d1,d0
000015AC           andi.b #$F0,d1          ;Grün-Bits*16 isolieren
000015B0           move.b d1,(a5)+         ;und eintragen
000015B2           andi.b #$0F,d0          ;Blau-Bits isolieren
000015B6           lsl.b #4,d0             ;Blauanteil*16
000015B8           move.b d0,(a5)+         ;eintragen
000015BA           addq.l #1,d7            ;Register erhöhen
000015BC           cmp.w d7,d5             ;nächstes Register lesen
000015BE           bhi farbloop1
000015C2           addq.l #8,d6            ;Farbzahl*3+8=Byte zu speichern
000015C4           move.l d6,d3
000015C6           bsr schreiben           ;Farbdaten schreiben
000015CA           beq saveend2            ;keine Daten vorhanden
000015CE           cmpi.l #-1,d0           ;Fehler?
000015D4           beq saveend2            ;ja, Ende

000015D8           move.l #"BODY",rpuff    ;Grafik-Bitmaps
000015E2           move.l d4,rpuff+4       ;Grafikgröße eintragen
000015E8           moveq.l #8,d3           ;8 Byte schreiben
000015EA           bsr schreiben
000015EE           move.l bildpointer,a0   ;Screenpointer holen
000015F4           move.l $58(a0),a5       ;Zeiger auf Bitmap aus Rastport
000015F8           clr.l d5                ;Zähler für Höhe
000015FA sa2       clr.l d6                ;Zähler für Tiefe
000015FC           moveq.l #8,d4           ;Zeiger auf erste Bitplane
000015FE sa1       move.l filepointer,d1   ;Zeiger auf Filepointer
00001604           move.l d5,d2            ;Zeile
```

```
00001606             addi.l #10,d2          ;Ab Zeile 10 wg. Titelleiste
0000160C             mulu widthd8,d2        ;* Breite/8 = Offset
00001612             add.l (a5,d4.l),d2     ;+ Startadresse Bitplane
00001616             clr.l d3
00001618             move.w widthd8,d3      ;Breite/8 Byte pro Zeile
0000161E             CALLDOS Write          ;Daten schreiben
00001628             beq saveend2           ;Keine Daten vorhanden
0000162C             cmpi.l #-1,d0          ;Fehler?
00001632             beq saveend2           ;Ja, Ende
00001636             addq.l #4,d4           ;Zeiger auf nächste Bitplane
00001638             addq.w #1,d6           ;Tiefe erhöhen
0000163A             cmp.w depth,d6
00001640             blt sa1                ;Nächste Bitplane
00001644             addq.w #1,d5           ;Höhe erhöhen
00001646             cmp.w heigth,d5
0000164C             blt sa2                ;Nächste Zeile lesen
00001650 saveend2    bsr close              ;Datei schließen
00001654 saveend1    bsr altdir             ;Altes Directory widerherstellen
00001658 saveend     rts
```

Zunächst muß der Kopf des IFF-Files, bestehend aus dem Text »FORM« und der Größe der Datei in Bytes gefolgt von dem Text »ILBM« als Kennzeichen für den Datenanfang, erzeugt werden. Die Größe der Datei setzt sich aus den Grafikbytes der Bitplanes, den Farbbytes (3 pro Farbe) sowie restlichen Angaben zusammen. Dabei wird die Größe des Kopfes selbst (8 Byte) nicht mitgerechnet. Die Anzahl der Grafikbytes haben wir schon des öfteren nach der Formel

Grafikbytes = (Screenbreite/8)*Screenhöhe*Screentiefe

berechnet. Die restlichen Bytes setzten sich so zusammen:

Text »ILBM«	: 4 Byte
BitMap-Header	:28 Byte
ColorMap	: 8 Byte (ohne Farbdaten)
Grafik	: 8 Byte (ohne Grafikbytes)
	48 Byte

Die Anzahl der Farben läßt sich durch die Formel 2 Screentiefe bestimmen. Dazu wird in einer Schleife, die von Null bis zur Screentiefe läuft, der Wert 1 jeweils um 1 Bit nach links verschoben, was einer Multiplikation mit dem Faktor 2 gleichkommt. Da pro Farbe die Anteile für Rot, Grün und Blau getrennt gespeichert werden müssen, sind also (Farbzahl*3) Bytes erforderlich, so daß sich die Dateigröße zu

Dateibytes = 48 + (Farbzehl*3) + (Screenbreite/8)*Screenhöhe*Screentiefe

ergibt. Danach wird der BitMap-Header mit Angaben zu den Screendimensionen erzeugt, die aus der Screen-Struktur ausgelesen werden.

Von der Screen-Höhe müssen wir natürlich wieder 10 subtrahieren, um die effiktive Grafikhöhe zu erhalten, nachdem wir sie beim Laden um 10 vergrößert hatten.

Als nächstes muß die ColorMap erzeugt werden. Das codierte Langwort aus der GetRGB4-Funktion wird durch logische Verknüpfungen in die Anteile der Grundfarben aufgespalten, wie wir es im letzten Kapitel besprochen haben, wobei allerdings zu beachten ist, daß die Werte erst noch mit 16 multipliziert werden müssen. Etwas unklar mögen die Verknüpfungen in der hexadezimalen Schreibweise schon sein, durch die binäre Schreibweise wird die Wirkung jedoch schnell deutlich:

Rotanteil*256 isolieren: Codelangwort and #$F00 (%0000111100000000)
Rotanteil*16 erzeugen mit Divison durch 16: lsr #4
Grünanteil*16 isolieren: Codelangwort and #$F0 (%0000000011110000)
Blauanteil isolieren: Codelangwort and #$0F (%0000000000001111)
Blauanteil*16 erzeugen durch Multiplikation mit 16: lsl #4

Die Speicherung der Grafikbytes schließlich vollzieht sich praktisch genauso, wie wir es beim Laden durchgeführt haben. Wenn Sie einmal die Assemblerzeilen vergleichen, werden Sie feststellen, daß sie fast von oben übernommen wurden, wobei die Daten hier natürlich den umgedrehten Weg aus der Bitplane in den Schreibpuffer nehmen. Nach diesen Informationen ist unsere Datei komplett, wir können sie also schließen und das Directory zurücksetzen.

5.3.6 Message holen und DOS-Unterprogramme

HiSoft GenAmiga Assembler 1.21 "IFF-Fileauswahl"

```
         * Gadget-ID aus Message-Port holen

0000165A message move.l 86(a0),a0          ;User-Port holen
0000165E         move.l a0,a5              ;Adresse retten
00001660         move.b 15(a0),d1          ;Signal-Bit holen
00001664         moveq #0,d0               ;Nummer in Maske
00001666         bset d1,d0                ;wandeln
00001668         CALLEXEC Wait             ;Auf Wiedersehen!
00001670         move.l a5,a0
00001672         CALLEXEC GetMsg           ;Message holen
0000167A         move.l d0,a1
0000167C         move.l $14(a1),d6         ;IDCMP-Flag holen
00001680         move.l $1C(a1),a4         ;Gadget-Adresse holen
00001684         CALLEXEC ReplyMsg         ;Message quittieren
0000168C         move.w $26(a4),d7         ;Gadget-ID holen
00001690         rts

         * Neues Directory setzen: (D2: -2=Lesen,-1=Schreiben)
```

```
00001692 newdir  move.l #Pfadname,d1         ;Zeiger auf Pfadnamen
00001698         CALLDOS Lock                ;Zeiger auf Lock-Struktur holen
000016A2         move.l d0,locksave          ;und merken
000016A8         beq endof                   ;nicht gefunden
000016AC         move.l d0,d1
000016AE         CALLDOS CurrentDir          ;Pfadname=neues Directory
000016B8         move.l d0,altpfad           ;alten Pfad merken
000016BE         move.l #111,d0              ;kein Fehler
000016C4 endof   rts


         * Altes Directory widerherstellen

000016C6 altdir  move.l altpfad,d1               ;Zeiger auf Lock-Struktur
000016CC         CALLDOS CurrentDir              ;altes Directory setzen
000016D6         rts


         * Datei öffnen  (D2: 1005=Lesen, 1006=Schreiben)

000016D8 open    move.l #Dateiname,d1            ;Zeiger auf Filenamen
000016DE         CALLDOS Open                    ;Datei öffnen
000016E8         move.l d0,filepointer           ;Pointer sichern
000016EE         rts


         * Datei schließen

000016F0 close   move.l filepointer,d1           ;Pointer holen
000016F6         CALLDOS Close                   ;Datei schließen
00001700         rts


         * Daten einlesen

00001702 lesen   move.l filepointer,d1           ;Pointer holen
00001708         move.l #rpuff,d2                ;Zeiger auf Datenpuffer
0000170E         CALLDOS Read                    ;Daten einlesen
00001718         rts

         * Daten schreiben

0000171A schreiben move.l filepointer,d1         ;Pointer holen
00001720           move.l #rpuff,d2              ;Zeiger auf Datenpuffer
00001726           CALLDOS Write                 ;Daten schreiben
00001730           rts
```

```
        * Screen des IFF-Bildes schließen, wenn erforderlich

00001732 clscreen  move.l bildpointer,d0          ;Screen geöffnet?
00001738           beq noscreen                   ;nein!
0000173C           move.l d0,a0
0000173E           CALLINT CloseScreen            ;ja, Screen schließen
00001748 noscreen  rts
```

Zu der Abfrage des Message-Ports muß man wohl wirklich nichts mehr sagen. Die Unterprogramme zur Bedienung der DOS-Funktionen sind wohl ebenfalls klar, lediglich das Schließen des IFF-Screens soll erwähnt werden. Hierfür wird der Screenpointer ausgelesen und auf den Wert Null überprüft. Die Null ist ein Kennzeichen dafür, daß noch kein Screen geöffnet wurde (Programmstartwert), somit kann man auf das Schließen des Screens natürlich verzichten. Ein Wert ungleich Null hingegen deutet darauf hin, daß von der Laderoutine ein Screen geöffnet wurde, der geschlossen werden muß.

5.3.7 Programmende und Strukturen

HiSoft GenAmiga Assembler 1.22 "IFF-Fileauswahl"

```
        *E N D E   P R O G R A M M


0000174A ende     bsr clscreen                    ;IFF-Screen schließen

0000174E          move.l windowpointer,a0         ;Window schließen
00001754          CALLINT CloseWindow

0000175E          move.l screenpointer,a0         ;Screen schließen
00001764          CALLINT CloseScreen

0000176E          move.l _DOSBase,a1              ;alle Librarys schließen
00001774          CALLEXEC CloseLibrary
0000177C fini3    move.l _GfxBase,a1
00001782          CALLEXEC CloseLibrary
0000178A fini2    move.l _IntuitionBase,a1
00001790          CALLEXEC CloseLibrary
00001798 fini1    rts

                  cnop 0,2

0000179A _DOSBase  dc.l 0                         ;Basis-Pointer-Libraries
0000179E _IntuitionBase  dc.l 0
000017A2 _GfxBase dc.l 0
```

```
000017A6 dosname  dc.b "dos.library",0           ;Namen-Libraries
                 cnop 0,2
000017B2 grafname  dc.b "graphics.library",0
000017C3          cnop 0,2
000017C4 intname  dc.b "intuition.library",0
                 cnop 0,2


000017D6 print             PRINTOUT text1,200,20
000017FC                   PRINTOUT text2,200,40
00001822                   PRINTOUT text3,200,60
00001848                   PRINTOUT text4,200,90
0000186E                   PRINTOUT text5,200,110
00001894                   PRINTOUT text6,200,130
000018BA                   rts


000018BC text1             TEXTOUT text11
000018D0 text2             TEXTOUT text22
000018E4 text3             TEXTOUT text33
000018F8 text4             TEXTOUT text44
0000190C text5             TEXTOUT text55
00001920 text6             TEXTOUT text66

                           even
00001934 text11            dc.b "***** Fileauswahl *****",0
                           even
0000194C text22            dc.b "      (w) 1988 by    ",0
                           even
00001962 text33            dc.b "   F.Riemenschneider",0
                           even
00001978 text44            dc.b "   Was wünschen Sie?",0
                           even
0000198E text55            dc.b "     Grafik LADEN    ",0
                           even
000019A4 text66            dc.b "   Grafik SPEICHERN",0
                           even


         * Strukturen

         * a) SCREENS

000019B8 screen_defs       SCREEN 640,400,$8000,titel
000019D8 bild              SCREEN 0,0,0,titel1

000019F8 titel   dc.b "IFF-Grafik",0
```

```
00001A03          cnop 0,4
00001A04 titel1   dc.b "IFF-Bild",0
00001A0D          cnop 0,4

00001A10 screenpointer  dc.l 0
00001A14 bildpointer    dc.l 0


         * b) WINDOWS

00001A18 waehlwindow  WINDOW
130,55,380,188,$60,igadget1,wwindowname,point1,$1006
00001A48 startwindow WINDOW0,20,639,235,$260,igadget111,swindowname,
         point2,$100E

00001A78 swindowname  dc.b   "IFF-Grafik laden/speichern",0

00001A93          cnop 0,2
00001A94 wwindowname  dc.b   "Fileauswahl",0
                  cnop 0,4

00001AA0 iwindowpointer  dc.l 0
00001AA4 windowpointer dc.l 0
                         even


         * b) GADGETS

         border
00001AA8           dc.w 0,0          ;x-/y-Offset
00001AAC           dc.b 3,1          ;Farben
00001AAE           dc.b   0          ;Farbmodus
00001AAF           dc.b   2          ;Zahl Koordinatenpaare
00001AB0           dc.l  koord1      ;Koordinatenpointer
00001AB4           dc.l   0          ;nächster Border

         koord1

00001AB8          dc.w     -2,10
00001ABC          dc.w     265,10

         border1
00001AC0           dc.w 0,0          ;x-/y-Offset
00001AC4           dc.b 3,1          ;Farben
00001AC6           dc.b   0          ;Farbmodus
00001AC7           dc.b   5          ;Zahl Koordinatenpaare
00001AC8           dc.l  koord2      ;Koordinatenpointer
00001ACC           dc.l   0          ;nächster Border
```

```
         koord2

00001AD0         dc.w     -2,-2
00001AD4         dc.w     77,-2
00001AD8         dc.w     77,10
00001ADC         dc.w     -2,10
00001AE0         dc.w     -2,-2


00001AE4 puffer  dc.w 0,0,0,0


         propinfo

00001AEC         dc.w  5         ;Vertikal verschieblich
00001AEE         dc.w 0,0        ;x,y-Position
00001AF2         dc.w 0          ;keine horizontale Verschiebung
00001AF4         dc.w $ffff      ;Schrittweite
00001AF6         dc.w 0          ;Breite
00001AF8         dc.w 0          ;Höhe
00001AFA         dc.w 0          ;Schrittweite horizontal
00001AFC         dc.w 0          ;Schrittweite vertikal
00001AFE         dc.w 0          ;Linker Rand
00001B00         dc.w 0          ;Rechter Rand


00001B02 string11       dc.b   "Pfadname :",0
                        even
00001B0E string22       dc.b   "Ausführen",0
                        even
00001B18 string33       dc.b   "Abbrechen",0
                        even
00001B22 string44       dc.b   "Dateiname :",0
                        even


00001B2E string1000     ds.b   33
                        even
00001B50 string1001     ds.b   33
                        even
00001B72 string1002     ds.b   33
                        even
00001B94 string1003     ds.b   33
                        even
00001BB6 string1004     ds.b   33
                        even
```

```
00001BD8 string1005      ds.b  33
                         even
00001BFA string1006      ds.b  33
                         even
00001C1C string1007      ds.b  33
                         even
00001C3E string1008      ds.b  33
                         even
00001C60 string1009      ds.b  33
                         even

00001C82 Dateiname       dc.b 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
00001C9A                 dc.b 0,0,0,0,0,0,0,0
                         even
00001CA2 Pfadname        dc.b "DF0:"
00001CA6                 ds.b 100
                         even

00001D0A string110       dc.b  "Laden",0
                         even
00001D12 string111       dc.b  "Speichern",0
                         even


00001D1C string1         STRING -90,0,string11,0
00001D30 string2         STRING 0,0,string22,0
00001D44 string3         STRING 0,0,string33,0
00001D58 string4         STRING -90,0,string44,0

00001D6C string100       STRING 0,0,string1000,0
00001D80 string101       STRING 0,0,string1001,0
00001D94 string102       STRING 0,0,string1002,0
00001DA8 string103       STRING 0,0,string1003,0
00001DBC string104       STRING 0,0,string1004,0
00001DD0 string105       STRING 0,0,string1005,0
00001DE4 string106       STRING 0,0,string1006,0
00001DF8 string107       STRING 0,0,string1007,0
00001E0C string108       STRING 0,0,string1008,0
00001E20 string109       STRING 0,0,string1009,0


00001E34 string20        STRING 0,0,string110,0
00001E48 string21        STRING 0,0,string111,0

         * Fileauswahl-Gadgets

00001E5C igadget1       GADGET igadget2,100,15,0,4,border,string1,strinfo1,0
```

```
00001E88 igadget2      GADGET igadget3,10,165,3,1,border1,string2,0,1
00001EB4 igadget3      GADGET igadget4,280,165,3,1,border1,string3,0,2
00001EE0 igadget4      PROPGADGET igadget5,348,35,3,3,puffer,0,propinfo,3
00001F0C igadget5      GADGET igadget6,10,35,0,1,0,string100,0,100
00001F38 igadget6      GADGET igadget7,10,45,0,1,0,string101,0,101
00001F64 igadget7      GADGET igadget8,10,55,0,1,0,string102,0,102
00001F90 igadget8      GADGET igadget9,10,65,0,1,0,string103,0,103
00001FBC igadget9      GADGET igadget10,10,75,0,1,0,string104,0,104
00001FE8 igadget10     GADGET igadget11,10,85,0,1,0,string105,0,105
00002014 igadget11     GADGET igadget12,10,95,0,1,0,string106,0,106
00002040 igadget12     GADGET igadget13,10,105,0,1,0,string107,0,107
0000206C igadget13     GADGET igadget14,10,115,0,1,0,string108,0,108
00002098 igadget14     GADGET igadget15,10,125,0,1,0,string109,0,109
000020C4 igadget15     GADGET 0,100,145,0,4,border,string4,strinfo2,4

000020F0 strinfo1      STRINF Pfadname
00002114 strinfo2      STRINF Dateiname


         * Wahl Laden/Speichern

00002138 igadget111     GADGET igadget112,250,170,3,1,boarder1,string20,0,0
00002164 igadget112     GADGET 0,250,200,3,1,border1,string21,0,1
```

Auch die Fileauswahl- sowie die IFF-Routinen bieten sich an, in Ihren eigenen Programmen verwendet zu werden. Zusammen mit den Informationen aus den vorigen Beispielprogrammen sollte es Ihnen gelingen, ein wirklich komfortabel zu bedienendes Programm in Assembler zu erstellen. Ich kann Ihnen aus eigener Erfahrung mitteilen, daß es sich wirklich lohnt, diese nicht nur zu Hause verstauben zu lassen, sondern an eine Fachzeitschrift wie z.B. das Amiga-Magazin zwecks einer Veröffentlichung zu schicken. Die Readakteure sind auch nur Menschen, wissen genausowenig über den Amiga wie Sie und ich und freuen sich besonders über Assembler-Programme. Sie können so mit Ihrem Hobby auch noch Ihre Finanzen aufbessern und vielleicht eines Tages selbst Buchautor werden. Für mich war die Mitarbeit an der Zeitschrift »64'er« das Sprungbrett zum Markt&Technik-Buchverlag.

5.4 Das Öffnen eines Fensters mit den DOS-Funktionen

Bisher habe ich Ihnen (mit Recht) verschwiegen, daß man außer von Intuition auch vom DOS aus ein Bildschirmfenster öffnen kann. Man benötigt dazu nicht einmal eine NewWindow-Struktur, dagegen kann man sogar durch die Verwendung von Steuerzeichen den Cursor bewegen, den Bildschirm scrollen, etc. Bevor Sie aber vor Wut über die Gemeinheit des Autors, Ihnen diese Informationen erst jetzt zu vermitteln, das Buch zer-

reißen, möchte ich Ihnen den entscheidenen Nachteil des DOS-Windows nahebringen: Ohne den direkten Zugriff auf das Console-Device (siehe nächstes Kapitel) kann man weder Gadgets noch Menüs benutzen, das Fenster weder vergrößern noch verkleinern, keine Fremdzeichensätze, keine Images, keine Borders verwenden, man muß also praktisch auf alle Spezialitäten des Amigas bzw. Intuitions verzichten, was für mich Grund genug war, das DOS-Fenster bislang links liegenzulassen.

Für gewisse Anwendungen, wie eine Textein- und ausgabe, kann man das Fenster aber durchaus benutzen. Geöffnet wird es mit der Open-Funktion. Dabei kann man zwischen den logischen Geräten CON: und RAW: wählen. Der einzige Unterschied besteht darin, daß Sie bei einem RAW:-Fenster die gesamte Tastatur einschließlich aller Sondertasten wie Funktions- oder Cursortasten benutzen können, woraus man schließen kann, daß es sich bei dem CLI-Window um ein CON:-Fenster handelt, da dort jede Wirkung der Cursortasten unterbleibt. Neben der Angabe des Gerätes muß man, jeweils durch Schrägstriche getrennt, die Koordinaten der linken oberen sowie der rechten unteren Ecke übergeben, wahlweise noch den Fenster-Namen:

```
filename  dc.b "CON:0/0/639/255/** Dies ist ein DOS-Fenster **",0
```

Interessant ist, daß die Fenster-Modi wie Interlace und HiRes automatisch abhängig von den Koordinaten gesetzt werden. Etwas merkwürdig ist die Tatsache, daß beim Öffnen des Fensters der Open-Modus mit Alt (1005) angegeben werden soll, da das Fenster ja noch nicht existiert. Nun ja, sehen wir es als kostenlose Zugabe zu den anderen schon behandelten Merkwürdigkeiten an.

Mit der Write-Funktion können Sie nun nach Herzenslust Zeichen in das Fenster ausgeben. Neben den »normalen« Zeichen existiert aber noch eine Reihe von Zeichen(sequenzen), die eine bestimmte Funktion auslösen, die sogenannten Steuerzeichen. Hier ein Überblick über die Wirkung der Steuerzeichen auf ein CON:- bzw. RAW:-Fenster.

Neben der Ausgabe können Sie aber auch Zeichen von der Tastatur mit Hilfe der Read-Funktion einlesen. Nach deren Aufruf erscheint der Cursor (wenn Sie ihn nicht unsichtbar gemacht haben), Sie können dann einen Text eingeben, der ab der angegebenen Adresse im Speicher abgelegt wird. Der wesentliche Unterschied zwischen einem CON:- und einem RAW:-Fenster bei der Eingabe besteht darin, daß in ersterem solange Text eingegeben werden kann, bis die Return-Taste gedrückt wird. In dem RAW:-Fenster hingegen wird die Eingabe schon nach dem ersten Druck einer Taste abgebrochen. Um dort längere Texte eingeben zu können, muß man daher die Read-Funktion wiederholt aufrufen und selbständig die Zeichen auf den Code für Return überprüfen. Während die »normalen« Zeichen im ASCII-Code abgelegt werden, bekommt man als Quittung für den Druck einer Sondertaste eine Sequenz aus 2, 3 oder 4 Byte zurück:

Sequenz	**Funktion**
08v	Backspace
0A	Linefeed, Cursor Down
0B	Cursor Up
0C	Fenster löschen
0D	Carriage Return
0E	Normaldarstellung (0F zurücksetzen)
0F	Auf Sonderzeichen schalten
9B <n> 40	n Leerzeichen einschieben
9B <n> 41	Cursor um <n> Zeilen hoch
9B <n> 42	Cursor um <n> Zeilen runter
9B <n> 43	Cursor um <n> Spalten rechts
9B <n> 44	Cursor um <n> Spalten links
9B <n> 45	Cursor um <n> Zeilen rauf, anschließend in Spalte 1
9B <n> 46	Cursor um <n> Zeilen runter, anschließend in Spalte 1
9B <n1> 3B <n2> 48	Cursor in Zeile <n1> und Spalte <n2> setzen
9B 4A	Fenster ab Cursor löschen
9B 4B	Zeile ab Cursor löschen
9B 4C	Zeile einfügen
9B 4D	Zeile löschen
9B <n> 50	<n>Zeichen ab Cursor löschen
9B <n> 53	<n>Zeilen hochschieben
9B <n> 54	<n>Zeilen runterschieben
9B 32 30 68	Linefeed mit Zeilenvorschub
9B 32 30 6C	Linefeed ohne Zeilenvorschub
9B 6E	Meldet Cursorposition : 9B <Zeile> 38 <Spalte> 52
9B <st><vf><hf> 6D	st: Stil des Zeichensatzens
	0 = Normal
	1 = Fettschrift
	3 = Italic
	4 = Unterstrichen
	7 = Invers
	vf : Vordergrundfarbe 30–37 entspricht Farbregister 0–7
	hf : Hintergrundfarbe 40–47 entspricht Farbregister 0–7
9B <n> 74	<n> = Maximale Anzahl der darstellbaren Zeilen
9B <n> 75	<n> = Maximale Zeilenlänge in Zeichen
9B <n> 78	<n> = Ausgabe vom linken Fensterrand in Pixeln
9B <n> 79	<n> = Ausgabe vom oberen Fensterrand in Pixeln
9B 30 20 70	Cursor sichtbar
9B 20 70	Cursor unsichtbar
9B 71	Sendet Fenster-Ausmaße in String mit folgendem
Aufbau:	
	9B 31 3B 31 3B <Zeilen> 3B <Spalten> 73

Taste	ohne Shift	mit Shift
F1	9B 30 7E	9B 31 30 7E
F2	9B 31 7E	9B 31 31 7E
F3	9B 32 7E	9B 31 32 7E
F4	9B 33 7E	9B 31 33 7E
F5	9B 34 7E	9B 31 34 7E
F6	9B 35 7E	9B 31 35 7E
F7	9B 36 7E	9B 31 36 7E
F8	9B 37 7E	9B 31 37 7E
F9	9B 38 7E	9B 31 38 7E
F10	9B 39 7E	9B 31 39 7E
HELP	9B 3F 7E	9B 3F 7E
Cursor Up	9B 41	9B 54 7E
Cursor Down	9B 42	9B 53 7E
Cursor Left	9B 43	9B 20 41 7E
Cursor Right	9B 44	9B 20 40 7E

5.5 CLI-Befehle vom DOS aufrufen

Sicherlich werden Sie sich auch schon manchmal gedacht haben, daß es in Ihren eigenen Programmen nützlich sein könnte, Befehle vom CLI aufrufen zu können. So könnte man das Directory einfach mit dem »DIR«-Befehl ausgeben, ohne umständlich mit der Lock-Funktion operieren zu müssen. Tatsächlich existiert in der DOS-Library eine Funktion mit dem Namen Execute, mit der man diesen Wunsch in die Tat umsetzen kann. Es ist dafür erforderlich, einen Zeiger auf einen String zu übergeben, der in seiner Syntax haargenau dem entsprechenden CLI-Befehl entspricht. Daneben müssen noch die File-Pointer der Ein- und Ausgabeeinheiten übergeben werden, mit denen der Befehl arbeiten soll. Um die Pointer zu erhalten, muß man zunächst die Dateien mit der Open-Funktion öffnen. Eine Null bedeutet, daß der Bildschirm angesprochen wird.

```
move.l #filename,d1      ;Zeiger auf Namen der Ausgabedatei
move.l #1006,d2          ;zum Schreiben öffnen
CALLDOS Open
move.l d0,d3             ;File-Pointer als Ausgabeeinheit
move.l #0,d2             ;keine Eingabedatei
move.l #befehl,d1        ;Zeiger auf Befehlsstring
CALLDOS Execute          ;CLI-Befehl aufrufen

filename    dc.b "df1:directory",0
            even
befehl      dc.b "dir df0:c",0
```

Mit diesem Mini-Programm würde das Directory des Ordners »c« des internen Laufwerks in eine Datei »directory« des externen Laufwerks gelistet. Falls Sie aber planen, Ihre Programme auch anderen Usern zugänglich zu machen, kann ich Ihnen nur von der Verwendung der Execute-Funktion abraten. Sie können nämlich nicht immer voraussetzen, daß der Fremdbenutzer auf seiner Diskette den Ordner mit den CLI-Befehlen parat hat. Zudem müssen Sie in jedem Fall davon ausgehen, daß er zumindest einige, recht selten benutzte Befehle aus dem Ordner gelöscht hat, um Speicherplatz zu gewinnen. In Programmen für den Eigenbedarf stellt die Execute-Funktion aber eine wertvolle Arbeitshilfe dar, mit der man Quelltext und Programm wesentlich kürzen kann.

Die Devices

Kapitel 6

Die Devices sind neben den Libraries das zweite wichtige Element für den Programmierer. Sie stellen die Verbindung zwischen der Software des Amiga und der Hardware dar, so daß man an sich von Treibern reden könnte. Dieser Ausdruck paßt hier aber nicht so recht, da die Devices eigenständige Tasks darstellen, die vom Hauptprogramm nicht mehr bedient werden müssen, wie dies bei konventionellen Treibern der Fall ist. Vielmehr wird wie bei Intuition über einen Message-Port zwischen den Devices und dem eigenen Programm kommuniziert. Der größte Vorteil der Devices liegt in deren Flexibilität. So sind sie größtenteils in dem Ordner »devs« auf der Workbench-Diskette gespeichert und können damit leicht an eine eventuelle neue Hardware angepaßt werden. Aber auch die ROM-devices können leicht geändert werden, ohne daß der Aufruf durch das Programm geändert werden müßte. Für alle I/O-Bereiche des Amiga existiert ein eigenes Device, insgesamt deren 17:

Devicename	**Steuerung von**
audio.device	Sound
clipboard.device	Datenübertragung
console.device	Ein-/Ausgabe über Bildschirm und Tastatur
gameport.device	Gameports 1 und 2
input.device	Gesamtsteuerung der Ein-/Ausgabe
inputevent.device	Ereignisse (z.B. Menüs, Gadgets, etc.)
jdisk.device	Sidecar oder PC-Karte beim Amiga 2000
keyboard.device	Tastaturzugriff
keymap.device	Tastaturbelegung
narrator.device	Sprachausgabe
parallel.device	Parallelport
printer.device	Drucker
prtbase.device	Datendefinition für printer.device
serial.device	serieller Port
timer.device	Systemzeit
trackdisk.device	Floppy

Das Vorgehen bei der Programmierung ist bei allen Devices identisch. Wir wollen als Beispiele die beiden zweifellos wichtigsten Devices, den trackdisk.device sowie den printer.device behandeln.

6.1 Allgemeine Programmierung eines Devices

Der Datenaustausch zwischen Programm und Device erfolgt wie gesagt über eine Message. Diese Message besteht aus der sogenannten I/O-Request-Struktur, in der alle wichtigen Daten für das Device eingetragen sind. Der Zeiger auf diese Struktur wird dem Device beim Aufruf übergeben. Hierfür bietet uns EXEC zwei Möglichkeiten an: Zum einen kann nach der Übergabe sofort in unserem Programm weitergemacht werden, da das Device als eigenständiger Task arbeitet. Es ist aber auch möglich, den eigenen Task bis zur Beendigung des Device-Tasks einzufrieren, was immer dann erforderlich ist, falls man unmittelbar auf die Device-Aktion reagieren möchte (z.B. deren Ergebnisse benötigt). Der letztere Weg ist aber auch aus einem anderen Grund zu empfehlen: Solange das Device mit einem I/O-Gerät beschäftigt ist, können Sie unmöglich aus Ihrem eigenen Task darauf zugreifen. Oft weiß man aber nicht, wie lange das Device beschäftigt sein wird und möchte selbst gerne Daten an dieses Gerät schicken, was natürlich erst nach der Beendigung des Device-Tasks möglich ist. Es ist also sicherlich kein Fehler zu warten.

Die Rückmeldung des Devices erfolgt über den Reply-Port, den wir zu Beginn selbst initialisieren müssen. Unter anderem müssen wir dort die Nummer unseres Taskes vermerken, damit er nach getaner Arbeit auch wieder aktiviert werden kann. Für diese kompliziert klingenden Vorgänge stellt uns Exec aber wieder fertige Unterprogramme in der Library zur Verfügung, so daß dies nicht weiter kompliziert ist.

Zunächst wollen wir den Reply-Port aktivieren. Dazu kann man die Funktion AddPort verwenden, der ein Zeiger auf die Port-Struktur übergeben werden muß:

```
move.l #portadr,a1          ;Zeiger auf Portstruktur
CALLEXEC AddPort            ;Port zu Systemports hinzufügen
```

Die Portstruktur ist wie folgt aufgebaut:

a) Node-Struktur
b) Message-Port-Flags
c) Signal-Bit
d) Zeiger auf Task
e) List-Struktur

Die Node-Strukur verbindet die einzelnen Ports in einer globalen Liste des Betriebssystems. Für uns sind die ersten beiden Langwörter unbedeutend, dann folgt ein Byte, das den Typ des Portes angibt. Da es sich um einen Message-Port handeln soll, müssen wir hier eine Vier eintragen (siehe Anhang). Dann folgt ein weiteres Byte, das die Priorität angibt. Hierauf kommen wir im nächsten Kapitel zu sprechen, wir können eine Null ein-

tragen. Als letztes folgt ein Langwort mit einem Zeiger auf den Port-Namen, z.B. »printer.port«, wenn wir das printer.device verwenden wollen. Da wir keine Flags zu verwenden brauchen, können wir auch das nächste Byte mit einer Null beschreiben. Das folgende Byte gibt die Nummer des sogenannten Singal-Bits an. Hierdurch wird später dem eigenen Task mitgeteilt, ob eine Message am Port angekommen ist. Sie kennen dieses Bit schon von Intuition. Bei Device-Zugriffen teilt das Bit mit, ob das Device seine Arbeit beendet hat. Hier können wir die Nummer des obersten Bits (31) einsetzen, da dieses praktisch immer unbenutzt ist.

Nun folgt der Zeiger auf den Task, um dessen Message-Port es sich handeln soll. Ihn können wir leicht mit der Exec-Funktion FindTask ermitteln und in die Struktur eintragen:

```
sub.l a1,a1             ;A1 löschen (eigenen Task ermitteln)
CALLEXEC FindTask       ;Zeiger auf Task holen
move.l d0,portadr+16    ;in Port-Struktur eintragen
```

Als letztes haben wir noch die List-Struktur, die aus vier Langwörtern und 2 Byte besteht, die von uns aber nicht beschrieben zu werden braucht. Damit hat die Port-Struktur insgesamt folgendes Aussehen:

```
portadr    dc.l  0
           dc.l  0
           dc.b  4     ;Kennzeichen Message-Port
           dc.b  0     ;Priorität
           dc.l  name  ;Zeiger auf Portnamen
           dc.b  0
           dc.b  31    ;Nummer Signalbit
           dc.l 0,0,0,0
           dc.b 0,0

name   "printer.port",0     ;z.B. bei Benutzung des Printer-Devices
```

Nachdem wir den Port mit der AddPort-Funktion eingerichtet haben, können wir das Device öffnen. Hierzu existiert eine weitere EXEC-Funktion mit dem Namen OpenDevice. Übergeben wird ihr ein Zeiger auf die I/O-Request-Struktur, den Device-Namen, die I/O-Einheit sowie diverse Flags, die man aber in der Regel zu Null setzen kann. Falls das Öffnen nicht geklappt hat, weil z.B. das Device nicht von der Diskette geladen werden konnte, wird eine Zahl ungleich Null zurückgegeben.

```
move.l #iorequest,a1     ;Zeiger auf I/O-Request-Struktur
move.l #devname,a0       ;Zeiger auf Devicenamen
move.l #1,d0             ;z.B. Disklaufwerk df1:
clr.l d1                 ;keine Flags
CALLEXEC OpenDevice      ;Device öffnen
tst.l d0
bne error                ;Fehler beim Öffnen

devname  dc.b "trackdisk.device",0 ;z.B. bei Benutzung des Trackdisk-Devices
```

Die I/O-Request-Struktur enthält alle wichtigen Daten für das Device, man kann sie als Message für das Device bezeichnen. So muß man z.B. dem trackdisk.device mitteilen, ob man von der Diskette Daten lesen oder eine Diskette formatieren möchte, dem printer.device wird u.a. die Größe des Ausdrucks mitgeteilt. Die Struktur besteht aus folgenden Elementen:

a) Message-Struktur
   - Node-Struktur
   - Zeiger auf Reply-Port
   - Länge der I/O-Request-Struktur
b) Zeiger auf Device
c) Zeiger auf Unit
d) Kommandowort
e) Device-spezifische Daten

Die Node-Struktur kennen wir bereits von der Port-Struktur. Auch hier muß nur das Typ-Byte beschrieben werden, hier mit einer 5 (siehe Anhang). In das Langwort für den Port-Zeiger tragen wir natürlich die Adresse der oben initialisierten Struktur ein. Dann folgt die Gesamtlänge der I/O-Request-Strukur in Bytes. Die folgenden beiden Langwörter werden von der OpenDevice-Funktion beschrieben, so daß wir hier eine Null eintragen können. Als letztes folgen dann die eigentlichen Informationen für jedes Device. Sie können sich sicherlich vorstellen, liebe Leser, daß sie von Device zu Device differieren, da ein trackdisk.device nun einmal andere Daten benötigt als ein printer.device. Auf diese speziellen Daten gehen wir in den folgenden Abschnitten ein, lassen Sie mich aber noch kurz den bisherigen Aufbau der I/O-Request-Struktur zusammenfassen:

```
iorequest   dc.l 0,0
            dc.b 5        ;Message-Typ
            dc.b 0
            dc.l 0
            dc.l portadr ;Zeiger auf Reply-Port
            dc.w größe    ;Struktur-Größe in Bytes (hängt von Device ab)
            dc.l 0,0
```

Die bisherige Länge beträgt 20 Byte. Nehmen wir einmal an, wir hätten auch alle devicespezifischen Daten in die Struktur eingetragen. Dann müssen wir als letztes das Device dazu bewegen, die Arbeit aufzunehmen. Dazu existieren zwei EXEC-Routinen. Die SendIO-Funktion startet das Device und kehrt sofort in den eigenen Task zurück, während die DoIO-Funktion diesen solange einfriert, bis das Device seine Arbeit beendet hat. Ich werde im folgenden immer die DoIO-Routine verwenden.

```
move.l #iorequest,a1             ;Zeiger auf I/O-Request Struktur
CALLEXEC DoIO (CALLEXEC SendIO)  ;Device starten
```

Falls Sie die SendIO-Methode gewählt haben, können Sie den Device-Task mit einer weiteren Funktion zu jeder Zeit abbrechen lassen:

```
move.l #iorequest,a1
CALLEXEC abortio
```

Nach Beendigung oder Abbruch des Devices müssen wir es wieder schließen und den Message-Port vom Betriebssystem abmelden. Hierzu kann man die folgenden Zeilen verwenden:

```
move.l #iorequest,a1
CALLEXEC Closedev        ;Device schließen
move.l #portadr,a1
CALLEXEC RemPort         ;Port abmelden
```

6.2 Das trackdisk.device

Dieses wichtige Device ist der Schlüssel zur gesamten Diskettenprogrammierung. So ist es nicht nur möglich, Fremdformate von z.B. IBM zu lesen, sondern auch Files zu retten, die mit den normalen DOS-Funktionen nicht mehr gelesen werden können. Bevor wir uns mit dem Aufbau dieser speziellen I/O-Struktur beschäftigen, müssen wir uns mit der Organisation der Diskettendaten auseinandersetzen.

Jede Diskette ist in die sogenannten Tracks (Ringe) aufgeteilt. Da die Amiga-Floppy zweiseitig arbeitet, liegen immer jeweils zwei Tracks übereinander. Diese Einheit wird Zylinder genannt. Über einem der Zylinder stehen die zwei Leseköpfe des Laufwerks, die die magnetischen Daten der Diskette in elektrische Impulse umwandeln. Um von einem Zylinder zum nächsten zu kommen, gibt es den sogenannten Steppermotor, der den Lesekopf in genau definierten Schritten über die Diskette hin und her bewegt.

Eine normale Amiga-Diskette besteht aus 80 Zylindern, wobei Zylinder 0 auf dem äußersten und Zylinder 79 auf dem innersten Ring liegt. Es existieren somit 160 Tracks, wobei die Tracks mit den ungeraden Nummern auf der Diskettenunterseite und die mit den geraden Nummern auf der Oberseite liegen. Der nullte Track liegt somit auf der oberen Seite des Zylinders Null, der letzte (159) auf der unteren Seite des Zylinders 79.

Ein Track wird wiederum in mehrere Sektoren (Blöcke) aufgeteilt. Beim Amiga liegen elf Sektoren auf einem Track. Der Sektor enthält nun die eigentlichen Daten, immer 512 Byte. Damit steht auch fest, warum unsere Amiga-Disketten eine Speicherkapazität von 880 Kbyte aufweisen:

80 Zylinder * 2 Tracks * 11 Sektoren * 512 Byte = 901 120 Byte = 880 Kbyte

Leider hat dieses Verfahren eine unangenehme Begleiterscheinung: Die Software zur Steuerung des Laufwerks muß mit drei Variablen (Seite, Track, Sektor) rechnen. Da hiermit zuviel Umstände verbunden wären, rechnet das trackdisk.device mit nur einer Variablen, der Blocknummer. Hierfür wurden alle 1760 Sektoren von 0 bis 1759 fortlaufend durchnumeriert. Die Blöcke 0 bis 10 befinden sich auf der oberen Diskettenseite im Track und Zylinder Null. Die nächsten elf Blöcke (11–21) liegen im gleichen Zylinder, aber auf

der gegenüberliegenden Seite im Track 1. Danach kommen die Tracks 2, 3, usw. an die Reihe. Diese Reihenfolge ist deshalb sinnvoll, weil der Schreib-Lesekopf nur einmal positioniert werden muß, um die Daten eines Zylinders zu lesen. Hätte man zunächst alle Blöcke der Oberseite und dann alle der Unterseite beschrieben, wären doppelt soviel Schritte erforderlich gewesen. Die Umrechnung in Blöcke geschieht nach der Formel

Blocknummer = 11*Track + Sektor

Die I/O-Request Struktur des trackdisk.devices hat folgende device-spezifischen Daten:

```
command    dc.w 0     ;Device-Kommando
           dc.w 0
IOerror    dc.l 0     ;s.u.
IOlenght   dc.l 0     ;s.u.
IOdata     dc.l 0     ;s.u.
IOoffset   dc.l 0     ;s.u.
```

Da nochmal 28 Byte dazugekommen sind, müssen Sie als Gesamtlänge eine 48 in die Struktur eintragen. Durch das Kommandowort wird festgelegt, was das trackdisk.device überhaupt tun soll. Es stehen hierfür unter anderen die folgenden Funktionen zur Verfügung:

Kommandowort	**Name**	**Funktion**
2	READ	esen von einem oder mehreren Sektoren
3	WRITE	Schreiben von einem bzw. mehreren Sekt.
4	UPDATE	Zurückschreiben des Track-Puffers
5	CLEAR	Track-Puffer löschen
9	MOTOR	Motor ein- oder ausschalten
10	SEEK	Suchen eines Tracks
11	FORMAT	Formatieren von Tracks
12	REMOVE	Disk-Test-Routine initialisieren
13	CHANGENUM	Anzahl der Diskettenwechsel feststellen
14	CHANGESTATE	Testen, ob Diskette eingelegt ist
15	PROTSTATUS	Diskette auf Schreibschutz testen

Beginnen wir mit den einfachsten Funktionen 13, 14 und 15, bei denen keine Parameter übergeben werden müssen. Nach der Ausführung des Device-Tasks steht in dem Langwort IOerror der Status zur Verfügung: Eine Null bedeutet, daß die Diskette nicht schreibgeschützt ist bzw. im Laufwerk liegt, eine 255 wird übergeben, wenn die Diskette schreibgeschützt ist oder nicht im Laufwerk liegt. Haben Sie das CHANGENUM-Kommando übergeben, steht an der gleichen Adresse die Zahl der Diskettenwechsel zur Verfügung.

Mit dem Kommando 9 wird der Laufwerksmotor, der dafür verantwortlich ist, daß sich die Diskette dreht, an- oder abgeschaltet. Dafür müssen Sie in das IOlenght-Langwort eine

Null (aus) oder eins (an) eintragen, bevor Sie das Device starten. Die Schreib- und Leseoperationen schalten sich übrigens bei Bedarf den Motor selbständig an, jedoch nicht wieder aus, so daß Sie hierfür diese Funktion benötigen. Das REMOVE-Kommando installiert eine Interrupt-Routine, die beim Entfernen der Diskette aus dem Laufwerk aufgerufen wird. Sie ist z.B. für das dauernde Klacken des Laufwerks verantwortlich, wenn keine Diskette eingelegt ist. Dafür muß der Zeiger auf die Interrupt-Routine in dem IOdata-Langwort übergeben werden. Falls eine Null eingetragen wird, wird die Interrupt-Routine abgeschaltet, was für die Nerven des Benutzers nicht das schlechteste sein soll. In Ihrem eigenen Interesse sollten Sie aber auf Eingriffe in diese Routine vermeiden, wenn Sie nicht genauestens über die Funktion des Diskcontrollers Bescheid wissen. Die READ- und WRITE-Funktionen arbeiten nach dem gleichen Muster. Dazu müssen neben dem Kommandowort übergeben werden:

- in IOlenght die Anzahl der zu lesenden/schreibenden Datenbytes
- in IOdata die Adresse des Lese/Schreibpuffers im Speicher
- in IOoffset die Datenadresse auf der Diskette

Letztere Adresse gibt das erste Byte auf der Diskette an, ab dem gelesen/geschrieben werden soll. Da jeder Block 512 Byte enthält, können Sie die Adresse nach der Formel

IOoffset = (Blocknummer - 1) *512 + Byte

ermitteln, wobei Byte von 0 bis 511 läuft und das Zugriffsbyte innerhalb des betrachteten Blocks angibt. Zu beachten ist, daß man im Speicher immer soviel Platz reservieren muß, daß auch alle Daten geladen werden können. Das Device nimmt keine Rücksicht darauf, ob hinter dem reservierten Bereich wichtige Programm- oder gar Betriebssystemdaten folgen, die einmal mehr den Guru ins Spiel bringen würden. Die folgende Routine ließt die Blöcke 4 bis 7 in den Speicher:

```
move.l #IOrequest,a1        ;Start IO-Request-Struktur
move.w #2,command           ;Kommandowort: Lesen
move.l #puffer,IOdata       ;Datenpuffer
move.l #4*512,IOlenght      ;Länge: 4 Blöcke
move.l #3*512,IOoffset      ;Offset für Lesebeginn
CALLEXEC DoIO               ;Device starten
move.w #9,command           ;Kommandowort: Motor ausschalten
CALLEXEC DoIO               ;Device starten

puffer   ds.b 512*4         ;Platz für Datenbytes
```

Wenn Sie einmal probieren, Daten auf eine (hoffentlich unwichtige) Diskette zu schreiben, werden Sie vielleicht feststellen, daß der Motor nicht anläuft. In der Tat werden die Daten aus Geschwindigkeitsgründen nur in einen RAM-Puffer geschrieben, dessen Größe Sie in der Startup-Sequenz mit dem Befehl AddBuffers vergrößern können. Dadurch wird die Zahl der benötigten Diskettenzugriffe reduziert, was mit einer höheren Verarbeitungsgeschwindigkeit identisch ist. Spätestens bevor man den Amiga auszuschalten gedenkt, sollte man die Daten aber auf die Diskette verewigen. Dazu dient das Kommando UPDATE,

welches ohne Parameter funktioniert. Der SEEK-Funktion müssen Sie lediglich den Offset übergeben, daraufhin wird der Schreib-Lese-Kopf in die entsprechende Position gebracht, ohne aber irgendeinen Zugriff auf die Diskette zu haben. Interessant ist dies für Programmierer von Kopierprogrammen, die eigene Lese- und Schreibroutinen verwenden.

Schließlich können Sie einen oder mehrere Tracks mit dem FORMAT-Kommando formatieren. Dazu müssen IOoffset, IOlänge und IOdata übergeben werden. Da man immer nur ganze Tracks formatieren kann, muß der Offset ein Vielfaches von 11*512 = 5632 sein, so würde man durch den Wert 2*5632 den Track 2 als ersten zu formatierenden festlegen. Aus dem gleichen Grund muß auch die Länge ein Vielfaches von 5632 sein. Schließlich muß das Langwort IOdata einen Zeiger auf den Speicherbereich beinhalten, in dem die Daten stehen, die auf den Track aufgetragen werden sollen. Bei mehreren zu formatierenden Tracks werden jeweils die gleichen Daten verwendet. Da der Aufbau eines Tracks relativ kompliziert ist (u.a. mit Syncmarkierungen, Prüfsummen, etc.) nützt es überhaupt nichtts, ihn mit Nullen zu beschreiben. Bei einem Zugriff auf diesen Track würden Sie außer Fehlermeldungen nichts bekommen.

6.3 Das printer.device

Das printer.device ist die Schnittstelle zum Drucker, wobei es keine Rolle spielt, ob Sie diesen über die parallele oder serielle Schnittstelle betreiben. Mit ihm ist es möglich, Steuerzeichen zu senden, Texte zu drucken oder Screen-Hardcopies anzufertigen.

Für jede dieser Funktionen steht eine eigene I/O-Struktur zur Verfügung. Beginnen wir mit dem Ausdruck einer Hardcopy.

Wahrscheinlich haben Sie sich schon wie ich über das völlig unzureichende Programm »GraphicDump« von der Workbench-Diskette geärgert, liebe Leser, da die Größe des Ausdrucks nicht zu beeinflussen und für mich viel zu klein ist. Dabei benutzt dieses Programm die gleiche Strategie wie unser gleich folgendes Beispielprogramm, das die gesamte Papierbreite ausnutzt und dazu noch eine Überschrift drucken kann. Sehen wir uns daher zunächst einmal den Aufbau der Hardcopy I/O-Request Struktur an:

```
IORequestH
command         dc.w 11    ;Kommandowort
                dc.b 0,0   ;keine Flags und I/O-Fehler
rast            dc.l 0     ;Zeiger auf Rastport
color           dc.l 0     ;Zeiger auf Color-Map
modes           dc.l 0     ;View-Modes
left            dc.w 0     ;Druckbeginn x-Koordinate (linker Rand)
up              dc.w 0     ;Druckbeginn y-Koordinate (oberer Rand)
width           dc.w 0     ;Breite auszugebener Bereich (indirekt rechter Rand)
heigth          dc.w 0     ;Höhe auszugebener Bereich (indirekt unterer Rand)
```

```
breite          dc.l 0    ;Druckbreite in Punkten
hoehe           dc.l 0    ;Druckhöhe in Punkten
flags           dc.w 0    ;Druckflags
```

Dieser Sruktur müssen Sie natürlich noch den allgemeinen Strukturbeginn (siehe oben) voraussetzen. Die Zeiger auf Rastport und Color-Map sind einfach aus der Screen-Strukur auszulesen. Auch die View-Modes stehen innerhalb der View-Port-Strukur in der Screen-Struktur, sind aber als Wort ab der Adresse $22 dort abgelegt. In der I/O-Request-Struktur ist jedoch ein Langwort erforderlich, so daß der 16-Bit-Wert auf 32 Bit ausgedehnt werden muß. In den folgenden vier Worten können Sie den gewünschten Bildschirmausschnitt definieren, der gedruckt werden soll. Im Zusammenspiel mit den folgenden beiden Langwörtern, die die Höhe und Breite des Ausdrucks angeben, können Sie Ihren individuellen Ausdruck definieren. Wenn Sie z.B. einen sehr kleinen Screen-Ausschnitt definieren, diesen aber in maximaler Größe (z.B. 960 Punkte Breite bei Epson-FX/RX) ausgeben lassen, werden Sie eine riesige Vergrößerung erhalten. Auf der anderen Seite können Sie natürlich auch den kompletten Screen auf Briefmarkengröße abbilden, indem Sie als Koordinatenursprung zweimal eine Null und als Breite sowie Höhe die Screen-Werte nehmen, die Druckhöhe und -Breite aber entsprechend klein wählen. Durch die Verwendung einiger Bit-Flags können Sie das Aussehen der Hardcopy noch zusätzlich beeinflussen:

Bit	Wert	Bedeutung
0	1	Festlegung von height in 1/1000
1	2	Festlegung von width in 1/1000
2	4	Height und width werden ignoriert, der Ausdruck wird in der maximal für den Drucker zulässigen Breite ausgeführt.
3	8	Height und width werden ignoriert, der Ausdruck wird in der maximal für den Drucker zulässigen Höhe ausgeführt.
4	16	Die Höhe ist ein Bruchteil von height.
5	32	Die Breite ist ein Bruchteil von width.
7	128	Height wird ignoriert. Die Höhe des Ausdrucks wird in Abhängigkeit der Breite festgelegt, so daß die Seitenverhältnisse mit denen des Screens übereinstimmen.
8	256	Geringst mögliche Punktdichte des Druckers wird verwendet.
9	512	Nächstgrößere Punktdichte wird verwendet.
8+9	768	Nächstgrößere Punktdichte wird verwendet.
10	1024	Höchstmögliche Punktdichte des Druckers wird verwendet.

Als Länge dieser Struktur müssen Sie den Wert 62 eintragen. Als nächstes haben wir die I/O-Struktur für die Übergabe von Druckersteuerzeichen. Sie ist entsprechend ihrer Funktion wesentlich kürzer als die der Hardcopy-Funktion, nämlich genau 38 Byte. Zu dem allgemeinen Strukturbeginn kommen noch folgende Daten:

```
IORequestS
command       dc.w 10      ;Kommandowort
              dc.b 0,0     ;Keine Flags und I/O-Fehler
prtcommand    dc.w 0       ;Druckerkommando
para0         dc.b 0       ;erster Parameter
para1         dc.b 0       ;zweiter Parameter
para2         dc.b 0       ;dritter Parameter
para3         dc.b 0       ;vierter Parameter
```

Eine Übersicht über alle Druckerkommandos und eventuell erforderliche Parameter finden Sie im Anhang dieses Buches. Natürlich ist es nicht möglich, alle Kommandos auf jedem Drucker einzusetzen, so wird z.B. das Kommandowort 25 (Schönschrift ein) auf einem Epson-RX 80 ohne Wirkung bleiben, da dieser Drucker keinen NLQ-Modus beherrscht.

Am einfachsten haben Sie es, wenn Sie einen Text ausgeben möchten. Hierfür können Sie nämlich die Struktur des trackdisk.devices übernehmen, wobei in dem Langwort IOdata ein Zeiger auf Ihren Text und in dem Langwort IOlenght dessen Länge zu übergeben ist. Wenn Sie hier eine Null eintragen, wird bis zum Ende des Textes gedruckt, d.h. bis zu dem ersten gefundenen Nullbyte. Als Kommandowort ist eine 9 zu übergeben.

In unserem nun folgenden Beispielprogramm wollen wir den Einsatz aller drei Möglichkeiten erproben. Hierzu soll eine Screen-Hardcopy in maximaler Größe gedruckt werden. Zunächst wollen wir mit der Text-Funktion eine Überschrift drucken. Dann wird mittels eines Steuerzeichens ein Wagenrücklauf sowie ein Zeilenvorschub durchgeführt. Als letztes wird schließlich der Screen-Dump ausgeführt. Als Breite wurden 960 Punkte festgelegt, durch Setzen des 7. Bit-Flags wurde die Höhe automatisch der Breite angepaßt. Das einzige Problem für uns besteht darin, die Startadresse der Screen-Struktur der Workbench zu erhalten. Hierzu müssen wir einen Trick anwenden: Durch Öffnen eines Windows auf der Workbench können wir die Adresse aus der Window-Struktur auslesen. Da wir das Window natürlich nicht mit auf unsere Hardcopy nehmen möchten, schließen wir es, bevor der Druck beginnt. Hier nun der Quelltext dieses Programms:

HiSoft GenAmiga Assembler 1.21

```
 "Printer-Device"
                opt l-,d+,s-,n+
                plen 72
                llen 80

                include df0:include/exec/exec_lib.i
                include df0:include/intuition/intuition_lib.i
                include df0:include/startup
```

```
0000006E begin   lea intname,a1            ;Intuition-Library öffnen
00000074         clr.l d0
00000076         CALLEXEC OpenLibrary
0000007E         tst.l d0
00000080         beq fini
00000084         move.l d0,_IntuitionBase

0000008A         lea newwindow,a0          ;Window öffnen
00000090         CALLINT OpenWindow
0000009A         move.l d0,windowpointer

000000A0         suba.l a1,a1
000000A2         CALLEXEC FindTask         ;eigenen Task ermitteln
000000AA         move.l d0,IOPort+16       ;in IOPort-Struktur eintragen
000000B0         lea IOPort,a1
000000B6         CALLEXEC AddPort

000000BE         lea IORequestT,a1         ;Zeiger auf IO Request-Struktur
000000C4         lea devicename,a0         ;Zeiger auf Devicenamen
000000CA         clr.l d0                  ;Unit 0
000000CC         clr.l d1                  ;keine Flags
000000CE         CALLEXEC OpenDevice       ;Device öffnen
000000D6         tst.l d0                  ;Fehler?
000000D8         bne error                 ;Ja, Abbruch!
000000DC         lea IORequestT,a1         ;Zeiger auf IO Request-Struktur
000000E2         CALLEXEC DoIO             ;Kommando ausführen
000000EA         lea IORequestT,a1
000000F0         CALLEXEC CloseDevice      ;und Device schließen

000000F8         lea IORequestS,a1         ;Zeiger auf IO Request-Struktur
000000FE         lea devicename,a0         ;Zeiger auf Devicenamen
00000104         clr.l d0                  ;Unit 0
00000106         clr.l d1                  ;keine Flags
00000108         CALLEXEC OpenDevice       ;Device öffnen
00000110         tst.l d0                  ;Fehler?
00000112         bne error                 ;Ja, Abbruch!
00000116         lea IORequestS,a1         ;Zeiger auf IORequest-Struktur
0000011C         CALLEXEC DoIO             ;Kommando ausführen
00000124         lea IORequestS,a1
0000012A         CALLEXEC CloseDevice      ;und Device schließen

00000132         move.l windowpointer,a0
00000138         move.l $2E(a0),a0         ;Adresse Screen-Struktur holen
0000013C         move.l a0,d0
0000013E         add.l #$54,d0             ;Zeiger auf Rast-Port-Struktur
00000144         move.l d0,rastp           ;in IORequest-Struktur
0000014A         move.w $0C(a0),width      ;Screen-Breite und -Höhe in
```

```
00000152         move.w $0E(a0),height     ;IORequest-Struktur eintragen
0000015A         adda.l #$2C,a0            ;Zeiger auf View-Port-Struktur
00000160         move.l 4(a0),color        ;Zeiger auf Color-Map und
00000168         clr.l d0
0000016A         move.w $20(a0),d0
0000016E         move.l d0,viewm           ;View-Mode in IORequest-Struktur
00000174         move.l windowpointer,a0   ;Window schliessen
0000017A         CALLINT CloseWindow

00000184         lea IORequestH,a1         ;Zeiger auf IORequest-Struktur
0000018A         lea devicename,a0         ;Zeiger auf Devicenamen
00000190         clr.l d0                  ;Unit 0
00000192         clr.l d1                  ;Keine Flags
00000194         CALLEXEC OpenDevice       ;Device öffnen
0000019C         tst.l d0                  ;Fehler?
0000019E         bne error                 ;Ja, Abbruch!
000001A2         lea IORequestH,a1         ;Zeiger auf IORequest-Struktur
000001A8         CALLEXEC DoIO             ;Kommando ausführen
000001B0         lea IORequestH,a1
000001B6         CALLEXEC CloseDevice      ;und Device schließen

000001BE error   lea IOPort,a1
000001C4         CALLEXEC RemPort
000001CC         move.l _IntuitionBase,a1  ;Library schließen
000001D2         CALLEXEC CloseLibrary
000001DA fini    rts

                 cnop 0,2

000001DC _IntuitionBase  dc.l 0
000001E0 intname  dc.b "intuition.library",0
                 cnop 0,2
000001F2 devicename  dc.b "printer.device",0
00000201         cnop 0,2
00000202 portname    dc.b "printer.port",0
0000020F         cnop 0,4

         * IOPort-Struktur für Messages

00000210 IOPort     dc.l 0,0        ;Reply-Port (Initialisierung im Programm)
00000218            dc.b 4
00000219            dc.b 0
0000021A            dc.l portname
0000021E            dc.b 0
0000021F            dc.b 31
00000220            dc.l 0
00000224 lh1        dc.l lh2
```

```
00000228 lh2        dc.l 0
0000022C            dc.l lh1
00000230            dc.b 0,0

00000232            cnop 0,4
         * IORequest-Struktur für Hardcopy

00000234 IORequestH dc.l 0,0
0000023C            dc.b 5,0
0000023E            dc.l 0
00000242            dc.l IOPort      ;Zeiger auf IOPort-Struktur
00000246            dc.w 62          ;Größe der IORequest-Struktur in Bytes
00000248            dc.l 0,0
00000250 command    dc.w 11          ;Kommando: DumpRastPort (Hardcopy)
00000252            dc.b 0,0
00000254 rastp      dc.l 0           ;Zeiger auf Rastport
00000258 color      dc.l 0           ;Zeiger auf Color-Map
0000025C viewm      dc.l 0           ;View-Port-Mode
00000260            dc.w 0           ;x-Ursprung
00000262            dc.w 0           ;y-Ursprung
00000264 width      dc.w 0           ;Breite des Rast-Ports
00000266 height     dc.w 0           ;Höhe des Rastports
00000268            dc.l 960         ;Breite des Ausdrucks in Punkten
0000026C            dc.l 0           ;Höhe des Ausdrucks in Punkten egal
00000270            dc.w 128         ;wegen Flag

00000272            cnop 0,4

         * IORequest-Struktur für Steuerzeichen

00000274 IORequestS dc.l 0,0
0000027C            dc.b 5,0
0000027E            dc.l 0
00000282            dc.l IOPort      ;Zeiger auf IOPort-Struktur
00000286            dc.w 48          ;Größe der IORequest-Struktur in Bytes
00000288            dc.l 0,0
00000290            dc.w 10          ;Kommando: Steuerzeichen senden
00000292            dc.b 0,0
00000294 steuer     dc.w 3           ;Code für Steuerzeichen
00000296 para1      dc.b 0           ;1.Parameter
00000297 para2      dc.b 0           ;2.Parameter
00000298 para3      dc.b 0           ;3.Parameter
00000299 para4      dc.b 0           ;4.Parameter
0000029A            dc.l 0,0,0,0
```

```
        * IORequest-Struktur für Textausgabe

000002AA IORequestT dc.l 0,0
000002B2            dc.b 5,0
000002B4            dc.l 0
000002B8            dc.l IOPort      ;Zeiger auf IO-Port-Struktur
000002BC            dc.w 48          ;Größe der IORequest-Struktur in Bytes
000002BE            dc.l 0,0
000002C6            dc.w 9           ;Kommando: Text drucken
000002C8            dc.b 0,0
000002CA            dc.l 0
000002CE long       dc.l -1          ;Textlänge
000002D2 print      dc.l text        ;Zeiger auf Text
000002D6            dc.w 0,0,0,0
000002DE            dc.l 0,0
000002E6            dc.w 0

                 cnop 0,4
000002E8 text    dc.b "******** Hardcopy des Workbenchscreens ********",13,0
                 even

        * NewWindow-Struktur

0000031E newwindow    dc.w        0      ;linke Ecke
00000320              dc.w        0      ;obere Ecke
00000322              dc.w        5      ;Breite
00000324              dc.w        5      ;Höhe
00000326              dc.b      0,1      ;Farben
00000328              dc.l        0      ;Flags für Events
0000032C              dc.l    $1000      ;Window Flags
00000330              dc.l        0      ;Zeiger auf erstes Gadget
00000334              dc.l        0      ;Check Mark
00000338              dc.l        0      ;Windowname
0000033C              dc.l        0      ;Screenpointer
00000340              dc.l        0      ;Bit Map
00000344              dc.w        0      ;Mindest-Breite
00000346              dc.w        0      ;Mindest-Höhe
00000348              dc.w        0      ;maximale Breite
0000034A              dc.w        0      ;maximale Höhe
0000034C              dc.w        1      ;Window-Typ (Workbench)

0000034E windowpointer  dc.l 0
```

Der Programmstart

Anhang 1

Auch bei der normalsten Sache des Computers, dem Programmstart, fällt unser Amiga völlig aus der Rolle. Er bietet uns die Möglichkeit, ein Programm entweder durch den Doppel-Mausklick von der Workbench oder durch die Eingabe des Namens vom CLI zu starten. Beide Startvorgänge laufen intern völlig unterschiedlich ab, so daß man bei Programmen, die sowohl von der Workbench als auch vom CLI startbar sein sollen, einen Programmteil installieren muß, der den Programmstart in die richtige Bahn lenkt: die Startup-Sequence. Bevor wir uns damit auseinandersetzen, wollen wir uns zunächst einmal ansehen, was denn überhaupt passiert, wenn wir ein Programm starten.

Wenn Sie sich einmal die CLI-Befehle im Ordner »c« ansehen, werden Sie feststellen, daß fast alle mit Parametern arbeiten. Die Frage, die sich einem stellt, ist die, woher das Programm (nichts anderes sind ja die CLI-Befehle) die Adresse des Parameterstrings und dessen Länge bekommt. Nun, freundlicherweise werden ihm diese Daten vom Amiga-Betriebssystem in den Registern D0 (Länge des Parameterstrings) und A0 (Adresse des Stringteils hinter dem Programmnamen) übergeben. Dadurch ist es ein leichtes, diese Parameter auszuwerten. Das folgende Mini-Programm z.B. stellt fest, welches Laufwerk (dfx:) als Parameter übergeben wurde:

```
tst.l d0          ;Parameterlänge
beq begin         ;0 !
cmp.l #5,d0       ;Länge dfx:+Return
bne begin
cmp.b #"d",(a0)   ;erstes Zeichen vergleichen
bne begin
cmp.b #"f",1(a0) ;zweites Zeichen vergleichen
bne begin
cmp.b #":",3(a0) ;viertes Zeichen vergleichen
bne begin
move.b 2(a0),d0  ;ASCII-Code von Laufwerksnummer
sub.b #48,d0     ;-ASCII-Code von Null = Laufwerksnummer
begin  ............
```

Im Gegensatz dazu wird bei einem Workbench-Start eine ganze Message übergeben. Selbst wenn man keine Lust hat, die dadurch übergebenen Informationen auszuwerten, muß man sie vom Message-Port abholen, um sie aus der Warteschlange der Messages zu entfernen.

Sonst würde man die Startup-Message nämlich an einer völlig ungeeigneten Stelle auslesen und wahrscheinlich damit den Vollstrecker in Form des Gurus auf den Plan rufen. Auf der anderen Seite darf man bei einem CLI-Start keine Message abholen, da diese ja nicht da ist! Aus dieser verzwickten Situation hilft uns die Struktur unseres Prozesses, deren Zeiger man mit der FindTask-Funktion erhalten kann und den wir ja schon bei den Devices benötigt haben. Ab der Adresse $AC steht dort ein Langwort, das immer dann Null ist, wenn das Programm von der Workbench gestartet wurde, bei einem CLI-Start jedoch ungleich Null ist.

Ein weiteres Problem beim Workbench-Start besteht darin, daß das Programm nicht einfach als Unterprogramm des jeweils schon aktivierten Tasks gestartet wird, wie dies beim CLI der Fall ist (Beweis: Das CLI nimmt keine Befehle an, solange Ihr Programm läuft), sondern als eigenständiger Task läuft. Bei einem Multitasking-Betriebssystem dürfen wir nicht einfach die bestehende Hackordnung der Tasks stören, indem wir einfach mit unserem Programm dazwischenschlagen, wir müssen vielmehr darauf warten, bis unser Task von EXEC die Starterlaubnis erhält, was durch das Anliegen der Message signalisiert wird (man stelle sich vor, auf einem Flughafen würde jedes Flugzeug sofort losfliegen, wenn es startbereit ist...). Da wir die Message noch benötigen, holen wir sie uns durch die GetMsg-Funktion vom Message-Port.

Nun können wir endlich unser eigenes Programm starten. Nach der regulären Beendigung (nach einer Guru-Meditation werden die folgenden Programmzeilen gar nicht mehr erreicht...) des Programms müssen wir unseren Task wieder ordnungsgemäß abmelden, sozusagen die Landeerlaubnis einholen. Dazu müssen wir die vorhin erhaltene Message durch die Reply-Funktion zurückgeben. Dies ist aber an dieser Stelle nicht ungefährlich, da auch die anderen Tasks Messages senden können. Um Verwirrungen und Gurus zu vermeiden, muß man für diesen Augenblick dem eigenen Task das alleinige Zugriffsrecht geben, wozu man die Forbid-Funktion benutzen kann. Sie schaltet alle weiteren Tasks solange ab, bis unser eigener Task beendet ist oder das Multitasking durch die Permit-Funktion wieder zugelassen wird. Da wir aber sowieso am Ende sind, können wir die Aktion mit dem RTS-Befehl beenden. Hier nun der Quellcode der Startup-Sequence:

```
*Startup-Sequence (fast) aus Amiga-Assembler-Buch
* ----------------------------------------------
         movem.l  d0/a0,-(sp)        ;Parameter retten
         clr.l  WBenchMsg            ;sicherheitshalber

* Teste, von wo wir gestartet wurden
* ---------------------------------
         sub.l  a1,a1                ;a1=0 = eigener Task
         CALLEXEC FindTask           ;Zeiger auf Task-Struktur
         move.l  d0,a4               ;Adresse retten
         tst.l  $AC(a4)              ;Laufen wir unter WB?
         beq.s  fromwb               ;ja
```

```
* Wir wurden vom CLI gestartet
* ----------------------------
         movem.l  (sp)+,d0/a0       ;Parameter holen
         bra  run                   ;und starten

* Wir wurden von  Workbench gestartet
* -----------------------------------
fromwb  lea  $5C(a4),a0             ;Zeiger auf Message-Port
        CALLEXEC WaitPort           ;Auf Start-Message warten
        lea  $5C(a4),a0
        CALLEXEC GetMsg             ;Message holen
        move.l  d0,WBenchMsg        ;und sichern!
        movem.l  (sp)+,d0/a0        ;Stackpointer korrigieren
run     bsr.s  begin                ;Programm aufrufen
        move.l  d0,-(sp)            ;Return-Code retten
        tst.l  WBenchMsg            ;WB-Message ungleich Null
        beq.s  exit                 ;nein, CLI-Start
        CALLEXEC Forbid             ;Tasks abschalten
        move.l  _WBenchMsg(pc),a1   ;Message holen
        CALLEXEC ReplyMsg           ;und zurückgeben
 exit   move.l  (sp)+,d0            ;Return-Code holen
        rts

WBenchMsg  ds.l 1
 cnop  0,2

begin ......                ;Start des eigenen Programms
```

Wenn Sie möchten, liebe Leser, können Sie die Startup-Message auch noch auswerten. Der in dem Label Wbenchmsg abgelegte Zeiger zeigt auf eine Struktur, die bis zur Adresse $14 mit der normalen Message-Struktur identisch ist, die wir von den Devices kennen. Dann folgen aber noch zusätzlich fünf Langworte, von denen besonders das dritte (ab Adresse $1C) und das fünfte (ab Adresse $24) interessant sind. Hierzu muß ein wenig weiter ausgeholt werden. Wie Sie sicherlich wissen, muß beim Start von der Workbench zusätzlich zu dem eigentlichen Programm noch ein weiteres File mit dem Programmnamen und dem Suffix .info existieren. Hiervon ist meistens aber nur bekannt, daß man diese Datei u.a. zur Icon-Darstellung benötigt. Nun, zu dem Aufbau der Info-Datei kommen wir gleich.

Zunächst wäre es wichtig zu wissen, wie man am einfachsten an diese Datei herankommt. Dazu existiert die Funktion mit dem Namen GetDiskObject, die sich in der Icon-Library befindet (die hatten wir noch gar nicht!). Der GetDiskObject-Funktion müssen wir im Register A0 einen Zeiger auf den Filenamen übergeben, zurück erhalten wir einen Zeiger auf die DiskObject-Struktur, die sich innerhalb der Datei befindet.

Den Zeiger auf den Filenamen erhalten wir schließlich aus einer Liste, auf die der Zeiger ab der Adresse $24 in der Message-Struktur gerichtet ist. Neben dem Zeiger auf den Filenamen enthält diese Liste noch den Datei-Lock. Dies ist wichtig, da sich das Programm theoretisch ja in einer Schublade befinden könnte und man daher erst mit der Current-Dir-Funktion das Directory entsprechend setzen müßte.

Immer wieder taucht die Frage auf, warum man zu diesen beiden Langwörtern Liste sagt. Nun, wie wir im Kapitel 2 gesehen haben, kann man auch gleichzeitig mehrere Icons anklicken. In diesem Fall stehen in der besagten Liste die Locks und Filenamenzeiger aller angeklickten Icons nacheinander zur Verfügung. Jetzt können Sie wahrscheinlich auch ahnen, wozu das Langwort ab der Adresse $1C in der Message-Struktur dient: Es enthält die Anzahl der angeklickten Icons.

Wenn wir dann endlich den Zeiger auf die DiskObject-Struktur erhalten haben, möchten wir natürlich auch kräftig in ihr herumpfuschen. Hierzu bieten sich besonders zwei Einträge an: Ab der Adresse $04 befindet sich die Gadget-Struktur des Icons. Hier können Sie Ihrer Phantasie freien Lauf lassen. So wirkt es sicherlich origineller, wenn die Schubladen-Icons der Unterdirectories geöffnet werden, falls man sie anklickt, als wenn sie einfach nur invertiert werden (siehe Diskette zum Buch).

Ab der Adresse $36 steht ein Langwort, das einen Zeiger auf eine weitere Zeigerliste darstellt. Diese Liste wird durch ein Null-Langwort abgeschlossen und zeigt auf die sogenannten Tool-Types. Hierbei handelt es sich einfach um Kommentartexte zum Programm, die ihrerseits wieder mit einem Nullbyte abgeschlossen werden. Hiermit kann man recht elegant Informationen an das Programm übergeben, so benutzt z.B. Notepad die Tool-Types für diverse Voreinstellungen des Textfensters. Man muß sich allerdings darüber im klaren sein, daß das Info-File durch die Tool-Types erheblich aufgebläht werden kann.

Durch die Funktion PutDiskObject, der Sie zwei Zeiger auf den Filenamen und die Struktur in den Registern A0 und A1 übergeben müssen, können Sie das geänderte Info-File wieder auf die Diskette zurückspeichern.

Die wichtigsten Funktionen und ihre Parameter

Anhang 2

1. **Exec**
2. **DOS**
3. **Intuition**
4. **Graphics**
5. **Layers**

1. Exec

Name	Register	Übergabeparameter
AbortIO	A1	IORequest
AddDevice	A1	Device
AddHead	A0,A1	List,Node
AddIntServer	A1,D0	Interrupt,IntNumber
AddLibrary	A1	Library
AddPort	A1	Port
AddResource	A1	Resource
AddTail	A1,A2	List,Node
AddTask	A1,A2,A3	Task,initPC,finalPC
AllocAbs	A1,D0	locatIOn,Size
Allocate	A0,D0	freeList,Size
AllocEntry	A0	Entry
AllocMem	D0,D1	Size,Request
AllocSignal	D0	SignalNum
AllocTrap	D0	TrapNum
AvailMem	D0	Request
Cause	A1	Interrupt
CheckIO	A1	IORequest
CloseDevice	A1	IORequest
CloseLibrary	A1	Library
Deallocate	A1,A2,D0	freeList,memoryBlock,Size
Disable	--	---
DoIO	A1	IORequest

Name	Register	Übergabeparameter
Enable	--	---
Enqueue	A0,A1	List,Node
FindName	A0,A1	List,name
FindPort	A1	name
FindTask	A1	name
Forbid	--	---
FreeEntry	A0	Entry
FreeMem	A1,D0	MemoryBlock,Size
FreeSignal	D0	SignalNum
FreeTrap	D0	TrapNum
GetMsg	A0	Port
Insert	A0,A1,A2	List,Node,pred
OldOpenLibrary	A1	LibName
OpenDevice	A0,A1,D0,D1	devName,IORequest,unit,Flags
OpenResource	A0,D0	resName,Version
Permit	--	---
PutMsg	A0,A1	Port,message
RemDevice	A1	Device
RemHead	A0	List
RemIntServer	A1,D0	Interrupt,IntNumber
RemLibrary	A1	Library
Remove	A1	Node
RemPort	A1	Port
RemResource	A1	Resource
RemTail	A0	List
RemTask	A1	Task
ReplyMsg	A1	Message
SendIO	A1	IORequest
SetExcept	D0,D1	NewSig,SignalSet
SetIntVector	A1,D0,D1	Interrupt,IntNum,Interrupt
SetSignal	D0,D1	NewSig,SignalSet
SetSR	D0,D1	NewSR,Mask
SetTaskPri	A1,D0	Task,Priority
Signal	A1,D0	Task,SignalSet
SumLibrary	A1	Library
SuperState	--	---
UserState	D0	SysStack
Wait	D0	SignalSet
WaitIO	A1	IORequest
WaitPort	A0	Port

2. DOS

Name	Register	Übergabeparameter
Close	D1	File
CreateDir	D1	Name
CreateProc	D1,D2,D3,D4	Name,Priorität,SegList,StackSize
CurrentDir	D1	Lock
DateStamp	D1	Date
Delay	D1	Timeout
DeleteFil	D1	Name
DeviceProc	D1	Name
DupLock	D1	Lock
Examine	D1,D2	Lock,FileInfoBLock
Execute	D1,D2,D3	String,File,File
Exit	D1	ReturnCode
ExNext	D1,D2	Lock,FileInfoBLock
Info	D1,D2	Lock,ParameterBLock
Input	--	---
IoErr	--	---
IsInteractive	D1	File
LoadSeg	D1	FileName
Lock	D1,D2	Name,Type
Open	D1,D2	Name,AccessMode
Output	--	---
ParentDir	D1	Lock
Read	D1,D2,D3	File,Buffer,Len
ReName	D1,D2	OldName,NewName
Seek	D1,D2,D3	File,Position,Offset
SetComment	D1,D2	Name,Comment
SetProtection	D1,D2	Name,Mask
UnLoadSeg	D1	Segment
UnLock	D1	Lock
WaitForChar	D1,D2	File,Timeout
Write	D1,D2,D3	File,Buffer,Len

3. Intuition

Name	Register	Übergabeparameter
AddGadget	A0,A1,D0	AddPtr,Gadget,Position
AllocRemember	A0,D0,D1	RememberKey,Sizef,Flags
AlohaWorkbench	A0	wbport
AutoRequest	A0,A1,A2,A3	Window,Body,PText,NText
	D0,D1,D2,D3	PFlag,NFlag,Width,Height
BeginRefresh	A0	Window
BuildSysRequest	A0,A1,A2,A3	Window,Body,PosText,NegText
	D0,D1,D2	Flags,Width,Height
ClearDMRequest	A0	Window
ClearMenuStrip	A0	Window
ClearPointer	A0	Window
CloseScreen	A0	Screen
CloseWindow	A0	Window
CloseWorkBench	--	---
CurrentTime	A0,A1	Seconds,Micros
DisplayAlert	D0,A0,D1	AlertNumber,String,Height
DisplayBeep	A0	Screen
DoubleClick	D0,D1,D2,D3	sseconds,smicros,cseconds,cmicros
DrawBorder	A0,A1,D0,D1	RPort,Border,LeftOffset,TopOffset
DrawImage	A0,A1,D0,D1	RPort,Image,LeftOffset,TopOffset
EndRefresh	A0,D0	Window,Complete
EndRequest	A0,A1	requester/Window
FreeRemember	A0,D0	RememberKey,ReallyForget
FreeSysRequest	A0	Window
GetDefPrefs	A0,D0	Preferences,Size
GetPrefs	A0,D0	Preferences/Size
InitRequester	A0	req
IntuiTextLength	A0	itext
Intuition	A0	ievent
ItemAddress	A0,D0	MenuStrip,MenuNumber
MakeScreen	A0	Screen
ModifyIDCMP	A0,D0	Window,Flags
ModifyProp	A0,A1,A2	Gadget/Ptr/Req
	D0,D1,D2,D3,D4	Flags,HPos,VPos,HBody,VBody
MoveScreen	A0,D0,D1	Screen,dx,dy
MoveWindow	A0,D0,D1	Window,dx,dy
OffGadget	A0,A1,A2	Gadget,Ptr,Req
OffMenu	A0,D0	Window,MenuNumber
OnGadget	A0,A1,A2	Gadget,Ptr,Req
OnMenu	A0,D0	Window,MenuNumber
OpenIntuition	--	---
OpenScreen	A0	OSargs
OpenWindow	A0	OWargs

Name	Register	Übergabeparameter
OpenWorkBench	--	---
PrintIText	A0,A1,D0,D1	rastport,itext,left,top
RefreshGadgets	A0,A1,A2	Gadgets/Ptr/Req
RemakeDisplay	--	---
RemoveGadget	A0,A1	RemPtr,Gadget
ReportMouse	A0,D0	Window,Boolean
Request	A0,A1	Requester,Window
RethinkDisplay	--	---
ScreenToBack	A0	Screen
ScreenToFront	A0	Screen
SetDMRequest	A0,A1	Window,req
SetMenuStrip	A0,A1	Window,Menu
SetPointer	A0,A1	Window,Pointer
	D0,D1,D2,D3	Height,Width,Xoffset,Yoffset
SetPrefs	A0,D0,D1	Preferences,Size,flag
SetWindowTitles	A0,A1,A2	Window,Windowtitle,Screentitle
ShowTitle	A0,D0	Screen,ShowIt
SizeWindow	A0,D0,D1	Window,dx,dy
ViewAddress	--	---
ViewPortAddress	A0	Window
WBenchToBack	--	---
WBenchToFront	--	---
WindowLimits	A0	Window
	D0,D1,D2,D3	minwidth,minheight, maxwidth, maxheight
WindowToBack	A0	Window
WindowToFront	A0	Window

4. Graphics

Name	Register	Übergabeparameter
AddAnimOb	A0,A1,A2	obj,animationKey,rastPort
AddBob	A0,A1	bob,rastPort
AddFont	A1	textFont
AddVSprite	A0,A1	vSprite,rastPort
AllocRaster	D0,D1	width,height
AndRectRegion	A0,A1	rgn,rect
Animate	A0,A1	animationKey,rastPort
AreaDraw	A1,D0,D1	rastPort,x,y
AreaEnd	A1	rastPort
AreaMove	A1,D0,D1	rastPort,x,y
AskFont	A1,A0	rastPort,textAttr
AskSoftStyle	A1	rastPort
BltBitMap	A0,A1,A2	srcBitMap,destBitMap,tempA
	D0,D1,D2, D3	srcX,srcY,destX,destY
	D4,D5,D6, D7	sizeX,sizeY,minterm,mask
BltBitMapRastPort	A0,A1	srcbm,destrp
	D0,D1,D2,D3	srcX,srcY,destX,destY
	D4,D5,D6	sizeX,sizeY,minterm
BltClear	A1,D0,D1	memory/size/flags
BltPattern	A1,A0	rastPort,ras
	D0,D1,D2,D3,D4	xl,yl,maxX,maxY,fillBytes
BltTemplate	A0,D0,D1,A1	src,srcX,srcMod,destRastPort
	D2,D3,D4,D5	destX,destY,sizeX,sizeY
CBump	A1	copperList
ChangeSprite	A0,A1,A2	vp,simplesprite,data
ClearEOL	A1	rastPort
ClearRegion	A0	rgn
ClearScreen	A1	rastPort
ClipBlit	A0,D0,D1,A1	src,srcX,srcY,destrp
	D2,D3,D4,D5,D6	destX,destY,sizeX,sizeY,minterm
CloseFont	A1	textFont
CMove	A1,D0,D1	copperList,destination,data
CopySBitMap	A0,A1	l1,l2
CWait	A1,D0,D1	copperList,x,y
DisownBlitter	--	---
DisposeRegion	A0	rgn
DoCollision	A1	rastPort
Draw	A1,D0,D1	rastPort,x,y
DrawGList	A1,A0	rastPort,viewPort
Flood	A1,D2,D0,D1	rastPort,mode,x,y
FreeColorMap	A0	colormap
FreeCopList	A0	coplist
FreeCprList	A0	cprlist

Name	Register	Übergabeparameter
FreeGBuffers	A0,A1,D0	animationObj,rastPort, doubleBuffer
FreeRaster	A0,D0,D1	planeptr,width,height
FreeSprite	D0	num
FreeVPortCopLists	A0	viewport
GelsFuncE	--	---
GelsFuncF	--	---
GetColorMap	D0	entries
GetGBuffers	A0,A1,D0	animationObj,rastPort, doubleBuffer
GetRGB4	A0,D0	colormap,entry
GetSprite	A0,D0	simplesprite,num
InitArea	A0,A1,D0	areaInfo,vectorTable, vectorTableSize
InitBitMap	A0,D0,D1,D2	bitMap,depth,width,height
InitGels	A0,A1,A2	dummyHead,dummyTail,GelsInfo
InitGMasks	A0	animationObj
InitMasks	A0	vSprite
InitRastPort	A1	rastPort
InitTmpRas	A0,A1,D0	tmpras,buff,size
InitView	A1	view
InitVPort	A0	viewPort
LoadRGB4	A0,A1,D0	viewPort,colors,count
LoadView	A1	view
LockLayerRom	A5	layer
MakeVPort	A0,A1	view,viewPort
Move	A1,D0,D1	rastPort,x,y
MoveSprite	A0,A1,D0,D1	viewport,simplesprite,x,y
MrgCop	A1	view
NewRegion	--	---
NotRegion	A0	rgn
OpenFont	A0	textAttr
OrRectRegion	A0,A1	rgn,rect
OwnBlitter	--	--
PolyDraw	A1,D0,A0	rastPort,count,polyTable
QBlit	A1	blit
QBSBlit	A1	blit
ReadPixel	A1,D0,D1	rastPort,x,y
RectFill	A1,D0,D1,D2,D3	rastPort,xl,yl,xu,yu
RemFont	A1	textFont
RemIBob	A0,A1,A2	bob,rastPort,viewPort
RemVSprite	A0	vSprite
ScrollRaster	A1,D0-D5	rastPort,dX,dY,minx,miny, maxx,maxy
ScrollVPort	A0	vp

Name	Register	Übergabeparameter
SetAPen	A1,D0	rastPort,pen
SetBPen	A1,D0	rastPort,pen
SetCollision	D0,A0,A1	type,routine,gelsInfo
SetDrMd	A1,D0	rastPort,drawMode
SetFont	A1,A0	RastPortID,textFont
SetRast	A1,D0	rastPort,color
SetRGB4	A0,D0,D1,D2,D3	viewPort,index,r,g,b
SetSoftStyle	A1,D0,D1	rastPort,style,enable
SortGList	A1	rastPort
SyncSBitMap	A0	l
Text	A1,A0,D0	RastPort,string,count
TextLength	A1,A0,D0	RastPort,string,count
UCopperListInit	A0,D0	copperlist,num
UnlockLayerRom	A5	layer
VBeamPos	--	---
WaitBlit	--	---
WaitBOVP	A0	viewport
WaitTOF	--	---
WritePixel	A1,D0,D1	rastPort,x,y
XorRectRegion	A0,A1	rgn,rect

5. Layers

Name	Register	Übergabeparameter
BeginUpdate	A0	Layer
BehindLayer	A0,A1	Layer info info,Layer
CreateBehindLayer	A0,A1,A2	Layer info info,bm,bm2
	D0,D1,D2,D3,D4	x0,y0,x1,y1,flags
CreateUpfrontLayer	A0,A1,A2	Layer info info,bm,bm2
	D0,D1,D2,D3,D4	x0,y0,x1,y1,flags
DeleteLayer	A0,A1	Layer info info,Layer
DisposeLayerInfo	A0	Layer info info
EndUpdate	A0,D0	Layer,flag
FattenLayerInfo	A0	Layer info info
InitLayers	A0	Layer info info
LockLayer	A0,A1	Layer
LockLayerInfo	A0	Layer info info
LockLayers	A0	Layer info info
MoveLayer	A0,A1,D0,D1	Layer info info,Layer,dx,dy
MoveLayerInFrontOf	A0,A1	Layer,Layer
NewLayerInfo	--	---
ScrollLayer	A0,A1,D0,D1	Layer info info,Layer,dx,dy
SizeLayer	A0,A1,D0,D1	Layer info info,Layer,dx,dy
SwapBitsRastPortCLayer	A0,A1	rp,cr
ThinLayerInfo	A0	Layer info info
UnlockLayer	A0	Layer
UnlockLayerInfo	A0	Layer info info
UnlockLayers	A0	Layer info info
UpfrontLayer	A0,A1	Layer info info,Layer
WhichLayer	A0,D0,D1	Layer info info,x,y

Library Vector Offsets

Anhang 3

Inhalt: _LVOs in alphabetischer Reihenfolge

1. Exec-Library

Öffnen: **exec.library**
Basis-Adresse: **_SysBase**

```
_LVOAbortIO          equ   -480
_LVOAddDevice        equ   -432
_LVOAddHead          equ   -240
_LVOAddIntServer     equ   -168
_LVOAddLibrary       equ   -396
_LVOAddPort          equ   -354
_LVOAddResource      equ   -486
_LVOAddTail          equ   -246
_LVOAddTask          equ   -282
_LVOAlert            equ   -108
_LVOAllocAbs         equ   -204
_LVOAllocate         equ   -186
_LVOAllocEntry       equ   -222
_LVOAllocMem         equ   -198
_LVOAllocTrap        equ   -342
_LVOAvailMem         equ   -216
_LVOCause            equ   -180
_LVOCheckIO          equ   -468
_LVOCloseDevice      equ   -450
_LVODeallocate       equ   -192
_LVODebug            equ   -114
_LVODisable          equ   -120
_LVODispatch         equ   -60
_LVODoIO             equ   -456
_LVOEnable           equ   -126
_LVOEnqueue          equ   -270
_LVOException        equ   -66
_LVOExitIntr         equ   -36
_LVOFindName         equ   -276
_LVOFindPort         equ   -390
_LVOFindResident     equ   -96
_LVOFindTask         equ   -294
_LVOForbid           equ   -132
_LVOFreeEntry        equ   -228
_LVOFreeMem          equ   -210
_LVOFreeSignal       equ   -336
_LVOFreeTrap         equ   -348
_LVOGetCC            equ   -528
_LVOGetMsg           equ   -372
_LVOInitCode         equ   -72
```

```
_LVOInitResident        equ   -102
_LVOInitStruct          equ   -78
_LVOInsert              equ   -234
_LVOMakeFunctions       equ   -90
_LVOMakeLibrary         equ   -84
_LVOOldOpenLibrary      equ   -408
_LVOOpenDevice          equ   -444
_LVOOpenLibrary         equ   -552
_LVOOpenResource        equ   -498
_LVOPermit              equ   -138
_LVOProcure             equ   -540
_LVOPutMsg              equ   -366
_LVORawDoFmt            equ   -522
_LVORawIOInit           equ   -504
_LVORawMayGetChar       equ   -510
_LVORawPutChar          equ   -516
_LVORemDevice           equ   -438
_LVORemHead             equ   -258
_LVORemIntServer        equ   -174
_LVORemLibrary          equ   -402
_LVORemove              equ   -252
_LVORemPort             equ   -360
_LVORemResource         equ   -492
_LVORemTail             equ   -264
_LVORemTask             equ   -288
_LVOReplyMsg            equ   -378
_LVOReschedule          equ   -48
_LVOSchedule            equ   -42
_LVOSendIO              equ   -462
_LVOSetExcept           equ   -312
_LVOSetFunction         equ   -420
_LVOSetIntVector        equ   -162
_LVOSetSignal           equ   -306
_LVOSetSR               equ   -144
_LVOSetTaskPri          equ   -300
_LVOSignal              equ   -324
_LVOAllocSignal         equ   -330
_LVOSumLibrary          equ   -426
_LVOSuperState          equ   -150
_LVOSupervisor          equ   -30
_LVOSwitch              equ   -54
_LVOTypeOfMem           equ   -534
_LVOUserState           equ   -156
_LVOVacate              equ   -546
_LVOWait                equ   -318
_LVOWaitIO              equ   -474
_LVOWaitPort            equ   -384
```

2. DOS-Library

Öffnen: **dos.library**
Basis-Adresse: **_DOSBase**

```
_LVOClose               equ   -36
_LVOCreateDir           equ   -120
_LVOCreateProc          equ   -138
_LVOCurrentDir          equ   -126
_LVODateStamp           equ   -192
_LVODelay               equ   -198
_LVODeleteFile          equ   -72
_LVODeviceProc          equ   -174
_LVODupLock             equ   -96
_LVOExamine             equ   -102
_LVOExecute             equ   -222
_LVOExit                equ   -144
_LVOExNext              equ   -108
_LVOGetPacket           equ   -162
_LVOInfo                equ   -114
_LVOInput               equ   -54
_LVOIoErr               equ   -132
_LVOIsInteractive       equ   -216
_LVOLoadSeg             equ   -150
_LVOLock                equ   -84
_LVOOpen                equ   -30
_LVOOutput              equ   -60
_LVOParentDir           equ   -210
_LVOQueuePacket         equ   -168
_LVORead                equ   -42
_LVORename              equ   -78
_LVOSeek                equ   -66
_LVOSetComment          equ   -180
_LVOSetProtection       equ   -186
_LVOUnLoadSeg           equ   -156
_LVOUnLock              equ   -90
_LVOWaitForChar         equ   -204
_LVOWrite               equ   -48
```

3. Intuition-Library

Öffnen: **intuition.library**
Basis-Adresse: **_IntuitionBase**

```
_LVOAddGadget          equ   -42
_LVOAllocRemember      equ   -396
_LVOAlohaWorkbench     equ   -402
_LVOAutoRequest        equ   -348
_LVOBeginRefresh       equ   -354
_LVOBuildSysRequest    equ   -360
_LVOClearDMRequest     equ   -48
_LVOClearMenuStrip     equ   -54
_LVOClearPointer       equ   -60
_LVOCloseScreen        equ   -66
_LVOCloseWindow        equ   -72
_LVOCloseWorkBench     equ   -78
_LVOCurrentTime        equ   -84
_LVODisplayAlert       equ   -90
_LVODisplayBeep        equ   -96
_LVODoubleClick        equ   -102
_LVODrawBorder         equ   -108
_LVODrawImage          equ   -114
_LVOEndRefresh         equ   -366
_LVOEndRequest         equ   -120
_LVOFreeRemember       equ   -408
_LVOFreeSysRequest     equ   -372
_LVOGetDefPrefs        equ   -126
_LVOGetPrefs           equ   -132
_LVOInitRequester      equ   -138
_LVOIntuition          equ   -36
_LVOItemAddress        equ   -144
_LVOLockIBase          equ   -414
_LVOMakeScreen         equ   -378
_LVOModifyIDCMP        equ   -150
_LVOModifyProp         equ   -156
_LVOMoveScreen         equ   -162
_LVOMoveWindow         equ   -168
_LVOOffGadget          equ   -174
_LVOOffMenu            equ   -180
_LVOOnGadget           equ   -186
_LVOOnMenu             equ   -192
_LVOOpenIntuition      equ   -30
_LVOOpenScreen         equ   -198
_LVOOpenWindow         equ   -204
_LVOOpenWorkBench      equ   -210
_LVOPrintIText         equ   -216
_LVORefreshGadgets     equ   -222
_LVORemakeDisplay      equ   -384
_LVORemoveGadget       equ   -228
_LVOReportMouse        equ   -234
_LVORequest            equ   -240
_LVORethinkDisplay     equ   -390
_LVOScreenToBack       equ   -246
_LVOScreenToFront      equ   -252
_LVOSetDMRequest       equ   -258
_LVOSetMenuStrip       equ   -264
_LVOSetPointer         equ   -270
_LVOSetPrefs           equ   -324
_LVOIntuiTextLength    equ   -330
_LVOSetWindowTitles    equ   -276
_LVOShowTitle          equ   -282
_LVOSizeWindow         equ   -288
_LVOUnlockIBase        equ   -420
_LVOViewAddress        equ   -294
_LVOViewPortAddress    equ   -300
_LVOWBenchToBack       equ   -336
_LVOWBenchToFront      equ   -342
_LVOWindowLimits       equ   -318
_LVOWindowToBack       equ   -306
_LVOWindowToFront      equ   -312
```

4. Graphics-Library

Öffnen: **graphics.library**
Basis-Adresse: **_GfxBase**

```
_LVOAddAnimOb          equ   -156
_LVOAddBob             equ   -96
_LVOAddFont            equ   -480
_LVOAddVSprite         equ   -102
_LVOAllocRaster        equ   -492
_LVOAndRectRegion      equ   -504
_LVOAnimate            equ   -162
_LVOAreaDraw           equ   -258
_LVOAreaEnd            equ   -264
_LVOAreaMove           equ   -252
_LVOAskFont            equ   -474
_LVOAskSoftStyle       equ   -84
_LVOBltBitMap          equ   -30
_LVOBltBitMapRastPort  equ   -606
_LVOBltClear           equ   -300
_LVOBltPattern         equ   -312
_LVOBltTemplate        equ   -36
_LVOCBump              equ   -366
```

```
_LVOChangeSprite        equ   -420
_LVOClearEOL            equ   -42
_LVOClearRegion         equ   -528
_LVOClearScreen         equ   -48
_LVOClipBlit            equ   -552
_LVOCloseFont           equ   -78
_LVOCMove               equ   -372
_LVOCopySBitMap         equ   -450
_LVOCWait               equ   -378
_LVODisownBlitter       equ   -462
_LVODisposeRegion       equ   -534
_LVODoCollision         equ   -108
_LVODraw                equ   -246
_LVODrawGList           equ   -114
_LVOFlood               equ   -330
_LVOFreeColorMap        equ   -576
_LVOFreeCopList         equ   -546
_LVOFreeCprList         equ   -564
_LVOFreeGBuffers        equ   -600
_LVOFreeRaster          equ   -498
_LVOFreeSprite          equ   -414
_LVOFreeVPortCopLists   equ   -540
_LVOGelsFuncE           equ   -180
_LVOGelsFuncF           equ   -186
_LVOGetColorMap         equ   -570
_LVOGetGBuffers         equ   -168
_LVOGetRGB4             equ   -582
_LVOGetSprite           equ   -408
_LVOInitArea            equ   -282
_LVOInitBitMap          equ   -390
_LVOInitGels            equ   -120
_LVOInitGMasks          equ   -174
_LVOInitMasks           equ   -126
_LVOInitRastPort        equ   -198
_LVOInitTmpRas          equ   -468
_LVOInitView            equ   -360
_LVOInitVPort           equ   -204
_LVOLoadRGB4            equ   -192
_LVOLoadView            equ   -222
_LVOLockLayerRom        equ   -432
_LVOMakeVPort           equ   -216
_LVOMove                equ   -240
_LVOMoveSprite          equ   -426
_LVOMrgCop              equ   -210
_LVONewRegion           equ   -516
_LVONotRegion           equ   -522
_LVOOpenFont            equ   -72
_LVOOrRectRegion        equ   -510
_LVOOwnBlitter          equ   -456
_LVOPolyDraw            equ   -336
_LVOQBlit               equ   -276
_LVOQBSBlit             equ   -294
_LVOReadPixel           equ   -318
_LVORectFill            equ   -306
_LVORemFont             equ   -486
_LVORemIBob             equ   -132
_LVORemVSprite          equ   -138
_LVOScrollRaster        equ   -396
_LVOScrollVPort         equ   -588
_LVOSetAPen             equ   -342
_LVOSetBPen             equ   -348
_LVOSetCollision        equ   -144
_LVOSetDrMd             equ   -354
_LVOSetFont             equ   -66
_LVOSetRast             equ   -234
_LVOSetRGB4             equ   -288
_LVOSetSoftStyle        equ   -90
_LVOSortGList           equ   -150
_LVOSyncSBitMap         equ   -444
_LVOText                equ   -60
_LVOTextLength          equ   -54
_LVOUCopperListInit     equ   -594
_LVOUnlockLayerRom      equ   -438
_LVOVBeamPos            equ   -384
_LVOWaitBlit            equ   -228
_LVOWaitBOVP            equ   -402
_LVOWaitTOF             equ   -270
_LVOWritePixel          equ   -324
_LVOXorRectRegion       equ   -558
```

5. Icon-Library

Öffnen: **icon.library**
Basis-Adresse: **_IconBase**

```
_LVOAddFreeList         equ   -72
_LVOAllocWBObject       equ   -66
_LVOBumpRevision        equ   -108
_LVOFindToolType        equ   -96
_LVOFreeDiskObject      equ   -90
_LVOFreeFreeList        equ   -54
_LVOFreeWBObject        equ   -60
_LVOGetDiskObject       equ   -78
_LVOGetIcon             equ   -42
```

```
_LVOGetWBObject        equ   -30
_LVOMatchToolValue     equ   -102
_LVOPutDiskObject      equ   -84
_LVOPutIcon            equ   -48
_LVOPutWBObject        equ   -36
```

6. Die drei Mathematik-Libraries

Öffnen: **mathffp.library**
Basis-Adresse: **_MathBase**

```
_LVOSPAbs              equ   -54
_LVOSPAdd              equ   -66
_LVOSPCmp              equ   -42
_LVOSPDiv              equ   -84
_LVOSPFix              equ   -30
_LVOSPFlt              equ   -36
_LVOSPMul              equ   -78
_LVOSPNeg              equ   -60
_LVOSPSub              equ   -72
_LVOSPTst              equ   -48
```

Öffnen: **mathieeedoubbas.library**
Basis-Adr.: **_MathIeeeDoubBasBase**

```
_LVOIEEEDPAbs          equ   -54
_LVOIEEEDPAdd          equ   -66
_LVOIEEEDPCmp          equ   -42
_LVOIEEEDPDiv          equ   -84
_LVOIEEEDPFix          equ   -30
_LVOIEEEDPFlt          equ   -36
_LVOIEEEDPMul          equ   -78
_LVOIEEEDPNeg          equ   -60
_LVOIEEEDPSub          equ   -72
_LVOIEEEDPTst          equ   -48
```

Öffnen: **mathtrans.library**
Basis-Adresse: **_MathTransBase**

```
_LVOSPAcos             equ   -120
_LVOSPAsin             equ   -114
_LVOSPAtan             equ   -30
_LVOSPCos              equ   -42
_LVOSPCosh             equ   -66
_LVOSPExp              equ   -78
_LVOSPFieee            equ   -108
_LVOSPLog              equ   -84
_LVOSPLog10            equ   -126
_LVOSPPow              equ   -90
_LVOSPSin              equ   -36
_LVOSPSincos           equ   -54
_LVOSPSinh             equ   -60
_LVOSPSqrt             equ   -96
_LVOSPTan              equ   -48
_LVOSPTanh             equ   -72
_LVOSPTieee            equ   -102
```

7. Sonstige

Öffnen: **diskfont.library**
Basis-Adresse: **DiskfontBase**

```
_LVOAvailFonts         equ   -36
_LVOOpenDiskFont       equ   -30
```

Öffnen: **translator.library**
Basis-Adresse: **TranslatorBase**

```
_LVOTranslate          equ   -30
```

Datentypen, Strukturen, Offset-Tabellen, Konstanten

Anhang 4

1. Exec

```
;node-Struktur
000             LN_SUCC ds.l    1       ; nächste Node
004             LN_PRED ds.l    1       ; vorhergehende Node
008             LN_TYPE ds.b    1       ;Node-Type (NT_...)
009             LN_PRI  ds.b    1       ;Priorität (meistens Null)
00A             LN_NAME ds.l    1       ;Name
                LN_SIZE equ     $00E    ;Strukturgröße in Bytes

NT_UNKNOWN      equ     0
NT_TASK         equ     1
NT_INTERRUPT    equ     2
NT_DEVICE       equ     3
NT_MSGPORT      equ     4
NT_MESSAGE      equ     5
NT_FREEMSG      equ     6
NT_REPLYMSG     equ     7
NT_RESOURCE     equ     8
NT_LIBRARY      equ     9
NT_MEMORY       equ     10
NT_SOFTINT      equ     11
NT_FONT         equ     12
NT_PROCESS      equ     13
NT_SEMAPHORE    equ     14
```

```
;Sys-Basis zählt ab hier

;lists-Struktur

000         LH_HEAD         ds.l    1         ;erste Node
004         LH_TAIL         ds.l    1         ;immer Null
008         LH_TAILPRED     ds.l    1         ;letzter gültiger Eintrag
00C         LH_TYPE         ds.b    1         ;Typ (siehe node-Typen)
00D         LH_pad          ds.b    1         ;unverwendet
            LH_SIZE         equ     $00E      ;Strukturgröße in Bytes

;Message-Port-Struktur

000-00D     MP_Node         ds.b    LN_SIZE   ;Node-Struktur
00E         MP_FLAGS        ds.b    1         ;Flags
00F         MP_SIGBIT       ds.b    1         ;Signal-Bit-Nummer
010         MP_SIGTASK      ds.l    1         ;^Task
014-021     MP_MSGLIST      ds.b    LH_SIZE   ;lists-Struktur
            MP_SIZE         equ     $022      ;Strukturgröße in Bytes

;Message-Struktur

000-00D     MN_Node         ds.b    LN_SIZE   ;Node-Struktur
00E         MN_REPLYPORT    ds.l    1         ;^Reply-Port
012         MN_LENGTH       ds.w    1         ;Länge in Bytes
014         MN_SIZE         equ     $014      ;Strukturgröße in Bytes

MP_SOFTINT       equ  MP_SIGTASK
PF_ACTION        equ  3

PA_SIGNAL        equ  0
PA_SOFTINT       equ  1
PA_IGNORE        equ  2

;Libraries

LIB_VECTSIZE     equ     6
LIB_RESERVED     equ     4
LIB_BASE         equ     $FFFFFFFA
LIB_USERDEF      equ     LIB_BASE-(LIB_RESERVED*LIB_VECTSIZE)
LIB_NONSTD       equ     LIB_USERDEF


000-00D     LIB_Node        ds.b    LN_SIZE   ;Node-Struktur
00E         LIB_FLAGS       ds.b    1         ;Flags
00F         LIB_pad         ds.b    1         ;unbenutzt
010         LIB_NEGSIZE     ds.w    1         ;Bereich von neg. Offsets
```

```
012         LIB_POSSIZE     ds.w    1           ;Bereich von Basisadresse
014         LIB_VERSION     ds.w    1           ;Library-Version
016         LIB_REVISION    ds.w    1           ;Anzahl Überarbeitung
018         LIB_IDSTRING    ds.l    1           ;^Name
01C         LIB_SUM         ds.l    1           ;Checksumme
020         LIB_OPENCNT     ds.w    1           ;Anzahl offene Tasks
            LIB_SIZE        equ     $022        ;Strukturgröße in Bytes

LIBB_SUMMING    equ     0                       ;Checksumme wird gerechnet
LIBF_SUMMING    equ     1                       ;
LIBB_CHANGED    equ     1                       ;Lib wurde geändert
LIBF_CHANGED    equ     2                       ;
LIBB_SUMUSED    equ     2                       ;1, wenn Checksum-Problem
LIBF_SUMUSED    equ     4
LIBB_DELEXP     equ     3
LIBF_DELEXP     equ     8

;Semaphore Message Port
022        SM_BIDS      ds.w   1                ;Anzahl Lock-Bits
           SM_SIZE      equ    $024

;Unions
SM_LOCKMSG      equ     MP_SIGTASK              ;siehe dort

;Device-Struktur
;Wie Library mit DD_xxxx
DD_SIZE    equ   $22

;Unit-Struktur
022         UNIT_FLAGS      ds.b    1
023         UNIT_pad        ds.b    1
024         UNIT_OPENCNT    ds.w    1
            UNIT_SIZE       equ     $022

 UNITB_ACTIVE    equ     0

;INTERRUPT-Struktur

000-00D     IS-Node         ds.b    LN-SIZE ;Node-Struktur
00E         IS_DATA         ds.l    1       ;Datenspeicher für Int
012         IS_CODE         ds.l    1       ;Interrupt-Programm
            IS_SIZE         equ     $016    ;Strukturgröße in Bytes

;INTERRUPT-Unterstruktur  (für jeden der 15 zulässigen Interrupts eine)

000         IV_DATA         ds.l    1       ;Datenspeicher für Int
004         IV_CODE         ds.l    1       ;Interrupt-Programm
```

```
008          IV_NODE        ds.l    1       ;Interrupt-Struktur
             IV_SIZE        equ     $00C    ;Strukturgröße in Bytes

SB_SAR          equu    15
SF_SAR          equ     $8000
SB_TQE          equ     14
SF_TQE          equ     $4000
SB_SINT         equ     13
SF_SINT         equ     $2000


SH_PAD          equ    SH_SIZE

SIH_PRIMASK     equ     $0F0
SIH_QUEUES      equ     5
```

Statische System-Variable

```
022          SoftVer        ds.w    1       ;Kickstart-Version
024          LowMemChkSum   ds.w    1       ;Checksumme Trap-Vekt.
026          ChkBase        ds.l    1       ;Sys-Basis (Komplement)
02A          ColdCapture    ds.l    1       ;^Cold-Start
02E          CoolCapture    ds.l    1       ;^Cool-Start
032          WarmCapture    ds.l    1       ;^Warm-Start
036          SysStkUpper    ds.l    1       ;^Sys-Stack (oben)
03A          SysStkLower    ds.l    1       ;^Sys-Stack (Top)
03E          MaxLocMem      ds.l    1       ;^akt. max. Memory
042          DebugEntry     ds.l    1       ;^Debugger
046          DebugData      ds.l    1       ;^Debugger Data-Segment
04A          AlertData      ds.l    1       ;^Alerts Data
04E          RsvdExt        ds.l    1       ;reserviert
052          ChkSum         ds.w    1       ;Checksumme bis hierher
;Für Interrupt-Quellen:
054          IntVects       equ     $054
054          IVTBE          ds.b    IV_SIZE
060          IVDSKBLK       ds.b    IV_SIZE
06C          IVSOFTINT      ds.b    IV_SIZE
078          IVPORTS        ds.b    IV_SIZE
084          IVCOPER        ds.b    IV_SIZE
090          IVVERTB        ds.b    IV_SIZE
09C          IVBLIT         ds.b    IV_SIZE
0A8          IVAUD0         ds.b    IV_SIZE
0B4          IVAUD1         ds.b    IV_SIZE
0C0          IVAUD2         ds.b    IV_SIZE
0CC          IVAUD3         ds.b    IV_SIZE
0D8          IVRBF          ds.b    IV_SIZE
0E4          IVDSKSYNC      ds.b    IV_SIZE
```

```
0F0         IVEXTER         ds.b    IV_SIZE
0FC         IVINTEN         ds.b    IV_SIZE
108         IVNMI           ds.b    IV_SIZE
;Dynamische System-Variable
114         ThisTask        ds.l    1    ;^aktueller Task
118         IdleCount       ds.l    1    ;Warte-Zähler
11C         DispCount       ds.l    1    ;Dispatch-Zähler
120         Quantum         ds.w    1    ;Zeit-Quantum
122         Elapsed         ds.w    1    ;davon verbraucht
124         SysFlags        ds.w    1    ;diverse
126         IDNestCnt       ds.b    1    ;Inter.-Disable-Tiefe
127         TDNestCnt       ds.b    1    ;Task-Disable-Tiefe

128         AttnFlags       ds.w    1    ;Merker-Flags
12A         AttnResched     ds.w    1    ;
12C         ResModules      ds.l    1    ;^residente Module

130         TaskTrapCode    ds.l    1    ;^Default Trap-Routine
134         TaskExceptCode  ds.l    1    ;^Dfault Exception-Rout.
138         TaskExitCode    ds.l    1    ;^Default Exit-Routine
13C         TaskSigAlloc    ds.l    1    ;Default Signal-Maske
140         TaskTrapAlloc   ds.w    1    ;Default Trap-Maske
;List Headers
142         MemList         ds.b    LH_SIZE
150         ResourceList    ds.b    LH_SIZE
15E         DeviceList      ds.b    LH_SIZE
16C         IntrList        ds.b    LH_SIZE
17A         LibList         ds.b    LH_SIZE
188         PortList        ds.b    LH_SIZE
196         TaskReady       ds.b    LH_SIZE
1A4         TaskWait        ds.b    LH_SIZE

1B2         SoftInts        ds.b    5*SH_SIZE
202         LastAlert       ds.b    16
212         ExecBaseReserved ds.l    1

SYSBASESIZE     equ     $216
;attention flags:
AFB_68010       equu     0      ;(auch 68020)
AFB_68020       equ      1
AFB_68881       equ      4
AFB_PAL         equ      8      ;PAL/NTSC
AFB_50HZ        equ      9      ;Clock-Rate

UNITF_ACTIVE    equ      1
UNITB_INTASK    equ      1
UNITF_INTASK    equ      2
```

```
;Memory-List-Struktur
000-00D     ML_Node          ds.b    LN_SIZE  ;Node-Struktur
00E         ML_NUMENTRIES    ds.w    1        ;Anzahl Speicherbereiche
010-018     ML_ME            ds.b    ME_SIZE  ;Memory-Entry-Struktur
            ML_SIZE          equ     24       ;Strukturgröße in Bytes

;Memory-Entry-Struktur
000         ME_REQS          ds.w    0        ;Bedingungs-Flags
000         ME_ADDR          ds.l    1        ;reservierender Speicher
004         ME_LENGTH        ds.l    1        ;Speichergröße
            ME_SIZE          equ     8        ;Strukturgröße in Bytes

MEMB_PUBLIC     equ     0
MEMF_PUBLIC     equ     1

MEMB_CHIP       equ     1
MEMF_CHIP       equ     2

MEMB_FAST       equ     2
MEMF_FAST       equ     4

MEMB_CLEAR      equ     16
MEMF_CLEAR      equ     $10000

MEMB_LARGEST    equ     17
MEMF_LARGEST    equ     $20000

MEM_BLOCKSIZE   equ 8
MEM_BLOCKMASK   equ (MEM_BLOCKSIZE-1)


;Memory-Header-Struktur
000-00D     MH_Node          ds.b    LN_SIZE   ;Node-Struktur
00E         MH_ATTRIBUTES    ds.w    1         ;Bedingungs-Flags
010         MH_FIRST         ds.l    1         ;erste MemChunk-Struktur
014         MH_LOWER         ds.l    1         ;Speicheranfang
018         MH_UPPER         ds.l    1         ;Speicherende
01C         MH_FREE          ds.l    1         ;erreichbare Speichergröße
            MH_SIZE          equ     $020      ;Strukturgröße in Bytes

;Memory-Chunk-Struktur
000         MC_NEXT          ds.l    1         ;nächste MemChunk-Struktur
004         MC_BYTES         ds.l    1         ;freie Speicherbytes
            MC_SIZE          equ     8         ;Strukturgröße in Bytes
```

```
;Task Control-Struktur

000-00D     TC_NODE         ds.b    LN_SIZE ;Node-Struktur
00E         TC_FLAGS        ds.b    1       ;Flags
00F         TC_STATE        ds.b    1       ;Status
010         TC_IDNESTCNT    ds.b    1       ;Interr. Disable-Tiefe
011         TC_TDNESTCNT    ds.b    1       ;Task-Disable-Tiefe
012         TC_SIGALLOC     ds.l    1       ;zugewiesene Signals
016         TC_SIGWAIT      ds.l    1       ;auf die gewartet wird
01A         TC_SIGRECVD     ds.l    1       ;die da sind
01E         TC_SIGEXCEPT    ds.l    1       ;die als Exception gelten
022         TC_TRAPALLOC    ds.w    1       ;zugewiesene Traps
024         TC_TRAPABLE     ds.w    1       ;Traps enabled
026         TC_EXCEPTDATA   ds.l    1       ;^Exception-Daten
02A         TC_EXCEPTCODE   ds.l    1       ;^Exeption-Code
02E         TC_TRAPDATA     ds.l    1       ;^Trap-Data
032         TC_TRAPCODE     ds.l    1       ;^Trap-Code
036         TC_SPREG        ds.l    1       ;Stack Pointer
03A         TC_SPLOWER      ds.l    1       ;^Stackframe unten
03E         TC_SPUPPER      ds.l    1       ;^           oben (+2)
042         TC_SWITCH       ds.l    1       ;^Task, der CPU verliert
046         TC_LAUNCH       ds.l    1       ;^Task, der dann kommt
04A-057     TC_MEMENTRY     ds.b    LH_SIZE ;Memory-Entry-Struktur
058         TC_Userdata     ds.l    1       ;^User-Daten
            TC_SIZE         equ     $05C    ;Strukturgröße in Bytes

TB_PROCTIME     equ     0
TF_PROCTIME     equ     1
TB_STACKCHK     equ     4
TF_STACKCHK     equ     16
TB_EXCEPT       equ     5
TF_EXCEPT       equ     32
TB_SWITCH       equ     6
TF_SWITCH       equ     64
TB_LAUNCH       equ     7
TF_LAUNCH       equ     128


TS_INVALID      equ     0
TS_ADDED        equ     TS_INVALID+1
TS_RUN          equ     TS_ADDED+1
TS_READY        equ     TS_RUN+1
TS_WAIT         equ     TS_READY+1
TS_EXCEPT       equ     TS_WAIT+1
TS_REMOVED      equ     TS_EXCEPT+1
```

```
SIGF_ABORT      equ   $0001
SIGF_CHILD      equ   $0002
SIGF_BLIT       equ   $0010
SIGF_DOS        equ   $0100

SIGB_ABORT      equ   0 SIGB_CHILD      equ   1
SIGB_BLIT       equ   4
SIGB_DOS        equ   8

SYS_SIGALLOC    equ   $0FFFF
SYS_TRAPALLOC   equ   $08000

;  IO-Request-Struktur

000-013      IO_MESSAGE      ds.b    MN_SIZE ;Message-Struktur
014          IO_DEVICE       ds.l    1       ;^Device-Struktur
018          IO_UNIT         ds.l    1       ;^Unit-Struktur
01C          IO_COMMAND      ds.w    1       ;^Device Command
01E          IO_FLAGS        ds.b    1       ;Flags
01F          IO_ERROR        ds.b    1       ;oder Warning-Code
             IO_SIZE         equ     $020    ;Strukturlänge in Bytes

;IO-Extension-Struktur

020          IO_ACTUAL       ds.l    1       ;Übertragene Bytes
024          IO_LENGTH       ds.l    1       ;Gesamt Bytes
028          IO_DATA         ds.l    1       ;^Daten
02C          IO_OFFSET       ds.l    1       ;Offset wenn Seeking
             IOSTD_SIZE      equ     $30     ;Strukturlänge
(Request+Extention)



IOB_QUICK       equ    0
IOF_QUICK       equ    1
```

2. DOS

```
;File-Zugriff
MODE_READWRITE  equ   1004
MODE_OLDFILE    equ   1005
MODE_READONLY   equ   MODE_OLDFILE
MODE_NEWFILE    equ   1006

;SEEK-Funktion, relative Positionen
OFFSET_BEGINNING  equ   -1                   ;Beginn des Files
```

```
OFFSET_BEGINING     equ   OFFSET_BEGINNING
OFFSET_CURRENT      equ    0                   ;relativ zu Ist
OFFSET_END          equ    1                   ;Ende des Files

;Trivialitäten
BITSPERBYTE         equ   8
BYTESPERLONG        equ   4
BITSPERLONG         equ   32
MAXINT              equ   $7FFFFFFF
MININT              equ   $80000000


;Arten von Locks
SHARED_LOCK         equ   -2                   ;andere Tasks dürfen lesen
ACCESS_READ         equ   SHARED_LOCK
EXCLUSIVE_LOCK      equ   -1                   ;dürfen nicht lesen
ACCESS_WRITE        equ   EXCLUSSIVE_LOCK

;DateStamp
00        ds_Days          ds.l    1           ;Tage seit 1.1.1978
04        ds_Minute        ds.l    1           ;Minuten seit 00 Uhr
08        ds_Tick          ds.l    1           ;Ticks in Minuten
          ds_SIZEOF        equ     $0C

TICKS_PER_SECOND          equ 50               ;1 Tick = 1/50 sec



;FileInfoBlock-Struktur
00      fib_DiskKey      ds.l    1            ;Diskettennummer
04      fib_DirEntryType ds.l    1            ;0=File, >0 = Dir.
08      fib_FileName     ds.b    108          ;Filename
74      fib_Protection   ds.l    1            ;Datei geschützt?
78      fib_EntryType    ds.l    1            ;Eintragstyp
7C      fib_Size         ds.l    1            ;Filegröße in Bytes
80      fib_NumBlocks    ds.l    1            ;Anzahl belegte Blocks
84-8F   fib_DateStamp    ds.b    ds_SIZEOF    ;Tage,Minuten,Ticks
90      fib_Comment      ds.b    116          ;Kommentar
        fib_SIZEOF       equ     $104         ;Strukturlänge in Bytes

FIBB_READ         equ     3
FIBF_READ         equ     8
FIBB_WRITE        equ     2
FIBF_WRITE        equ     4
```

```
FIBB_EXECUTE     equ     1
FIBF_EXECUTE     equ     2
FIBB_DELETE      equ     0
FIBF_DELETE      equ     1

;InfoData-Struktur
;muß auf eine Langwortgrenze justiert sein!

                 CNOP                0,4
00               id_NumSoftErrors    ds.l    1    ;Anzahl Diskettenfehler
04               id_UnitNumber       ds.l    1    ;installierte Diskeinheit
08               id_DiskState        ds.l    1    ;Diskettenstatus
0C               id_NumBlocks        ds.l    1    ;Anzahl Sektoren einer Disk
10               id_NumBlocksUsed    ds.l    1    ;Anzahl verwendete Sektoren
14               id_BytesPerBlock    ds.l    1    ;Anzahl Bytes pro Sektor
18               id_DiskType         ds.l    1    ;Diskettentyp
1C               id_VolumeNode       ds.l    1    ;Diskettenname
20               id_InUse            ds.l    1    ;<>0, falls Diskette aktiv
                 id_SIZEOF           equ  $24     ;Strukturgröße in Bytes

ID_WRITE_PROTECTED   equ   80
ID_VALIDATING        equ   81
ID_VALIDATED         equ   82
ID_NO_DISK_PRESENT   equ   -1
ID_UNREADABLE_DISK   equ     $42414400        ; 'BAD'  geshifted
ID_NOT_REALLY_DOS    equ     $E444F53         ; 'NDOS'
ID_DOS_DISK          equ     $44F5300         ; 'DOS'
ID_KICKSTART_DISK    equ     $B49434B         ; 'KICK'

;Error-Codes
ERROR_NO_FREE_STORE            equ  103
ERROR_OBJECT_IN_USE            equ  202
ERROR_OBJECT_EXISTS            equ  203
ERROR_OBJECT_NOT_FOUND         equ  205
ERROR_ACTION_NOT_KNOWN         equ  209
ERROR_INVALID_COMPONENT_NAME   equ  210
ERROR_INVALID_LOCK             equ  211
ERROR_OBJECT_WRONG_TYPE        equ  212
ERROR_DISK_NOT_VALIDATED       equ  213
ERROR_DISK_WRITE_PROTECTED     equ  214
ERROR_RENAME_ACROSS_DEVICES    equ  215
ERROR_DIRECTORY_NOT_EMPTY      equ  216
ERROR_DEVICE_NOT_MOUNTED       equ  218
ERROR_SEEK_ERROR               equ  219
ERROR_COMMENT_TOO_BIG          equ  220
ERROR_DISK_FULL                equ  221
ERROR_DELETE_PROTECTED         equ  222
```

```
ERROR_WRITE_PROTECTED         equ  223
ERROR_READ_PROTECTED          equ  224
ERROR_NOT_A_DOS_DISK          equ  225
ERROR_NO_DISK                 equ  226
ERROR_NO_MORE_ENTRIES         equ  232

;empfohlene Return-Codes
RETURN_OK       equ    0       ;kein Fehler
RETURN_WARN     equ    5       ;nur Warnung/Hinweis
RETURN_ERROR    equ   10       ;Fehler
RETURN_FAIL     equ   20       ;Totalschaden

;Break-Codes
SIGBREAKB_CTRL_C        equ     12
SIGBREAKF_CTRL_C        equ     $1000
SIGBREAKB_CTRL_D        equ     13
SIGBREAKF_CTRL_D        equ     $2000
SIGBREAKB_CTRL_E        equ     14
SIGBREAKF_CTRL_E        equ     $4000
SIGBREAKB_CTRL_F        equ     15
SIGBREAKF_CTRL_F        equ     $8000
```

3. Intuition

```
;Menu-Struktur

00             mu_NextMenu     ds.l    1      ;^nächstes Menü
04             mu_LeftEdge     ds.w    1      ;linke Ecke
06             mu_TopEdge      ds.w    1      ;obere Ecke
08             mu_Width        ds.w    1      ;Breite
0A             mu_Height       ds.w    1      ;Höhe
0C             mu_Flags        ds.w    1      ;Menü wählbar?
0E             mu_MenuName     ds.l    1      ;^Titel-Text
12             mu_FirstItem    ds.l    1      ;^erster Menüpunkt
16             mu_JazzX        ds.w    1      ;intern
18             mu_JazzY        ds.w    1
1A             mu_BeatX        ds.w    1
1C             mu_BeatY        ds.w    1
               mu_SIZEOF       equ     $1E    ;Strukturlänge in Bytes

MENUENABLED  equ $0001           ;aktiv
MIDRAWN   equ $0100              ;gezeichnet

;Menüpunkt-Struktur

00             mi_NextItem     ds.l    1      ;^nächster Menüpunkt
04             mi_LeftEdge     ds.w    1      ;linke Ecke
```

```
06              mi_TopEdge       ds.w    1      ;obere Ecke
08              mi_Width         ds.w    1      ;Breite
0A              mi_Height        ds.w    1      ;Höhe
0C              mi_Flags         ds.w    1      ;Flags
0E              mi_MutualExclude ds.l    1      ;gegenseitiger Ausschluß
12              mi_ItemFill      ds.l    1      ;^Text oder ^Image
16              mi_SelectFill    ds.l    1      ;Image bei Berührung
1A              mi_Command       ds.b    1      ;Anwahl auch durch Taste
1B              mi_AdjustToWord  ds.b    1      ;Wortgrenze
1C              mi_SubItem       ds.l    1      ;^Untermenü-Struktur
20              mi_NextSelect    ds.w    1      ;nächster Menüpunkt
                mi_SIZEOF        equ     $22    ;Strukturlänge in Bytes

CHECKIT          equ $0001     ;Menüpunkt bei Anwahl abhaken
ITEMTEXT         equ $0002     ;Menüpunkt hat Text sonst Image
COMMSEQ          equ $0004     ;Anwahl auch durch Amiga-Taste
MENUTOGGLE       equ $0008     ;Wechselweise abhaken
ITEMENABLED      equ $0010     ;Menüpunkt unwählbar
HIGHIMAGE        equ $0000     ;alternativ Image/Text
HIGHCOMP         equ $0040     ;bei Berührung Menüpunkt invertieren
HIGHBOX          equ $0080     ;bei Berührung Box umrahmen
HIGHNONE         equ $00C0     ;bei Berührung keine Reaktion
CHECKED          equ $0100     ;1 wenn Menüpunkt abgehakt
ISDRAWN          equ $1000     ;1 wenn Menüpunkt auf Schirm
HIGHITEM         equ $2000     ;1 wenn Reaktion auf Berührung
MENUTOGGLED      equ $4000     ;1 wenn bei Berührung umgeschaltet

NOMENU           equ  $001F    ;1/0= Menü an/aus
NOITEM           equ  $003F    ;     Menüpunkt an/aus
NOSUB            equ  $001F    ;     Untermenüpunkt an/aus
MENUNULL         equ  $FFFF    ;Kein Menüpunkt angewählt
CHECKWIDTH       equ  19       ;Platz für Check-Marke
COMMWIDTH        equ  27       ;Platz für Taste wenn HighRes
LOWCHECKWIDTH    equ  13       ;wenn niedrige Auflösung


;Requester-Struktur

00              rq_OlderRequest ds.l    1      ; voriges Requester
04              rq_LeftEdge     ds.w    1      ;linke Ecke
06              rq_TopEdge      ds.w    1      ;obere Ecke
08              rq_Width        ds.w    1      ;Breite
0A              rq_Height       ds.w    1      ;Höhe
0C              rq_RelLeft      ds.w    1      ;wenn Pointer
0E              rq_RelTop       ds.w    1      ;Bezugspunkt
10              rq_ReqGadget    ds.l    1      ;^Gadget-Liste
14              rq_ReqBorder    ds.l    1      ;^Border-Struktur
```

```
18          rq_ReqText      ds.l    1     ;^Text-Struktur
1C          rq_Flags        ds.w    1     ;Bits siehe unten
1E          rq_BackFill     ds.b    1     ;Hintergrund-Farbe
1F          rq_AdjustToWord ds.b    1
20          rq_ReqLayer     ds.l    1     ;^Layer-Struktur
24          rq_ReqPad1      ds.b    32    ;reserviert
44          rq_ReqBMap      ds.l    1     ;^Custom Bit Map
48          rq_RWindow      ds.l    1     ;reserviert
4C          rq_ReqPad2      ds.b    36    ;reserviert
            rq_SIZEOF       equ     $70   ;Strukturgröße in Bytes

POINTREL        equ $0001    ;1 wenn relativ zu Mauszeiger
PREDRAWN        equ $0002    ;1 wenn Custom BitMap
REQOFFWINDOW    equ $1000    ;wenn Requester außerhalb Window
REQACTIVE       equ $2000    ;0/1= Requester aktiv
SYSREQUEST      equ $4000    ;nur wenn System-Requester
DEFERREFRESH    equ $8000    ;

;Gadget-Struktur

00          gg_NextGadget   ds.l    1     ;^nächstes Gadget
04          gg_LeftEdge     ds.w    1     ;linke Ecke
06          gg_TopEdge      ds.w    1     ;obere Ecke
08          gg_Width        ds.w    1     ;Breite
0A          gg_Height       ds.w    1     ;Höhe
0C          gg_Flags        ds.w    1     ;Flags (Reaktion auf Anklicken)
0E          gg_Actiation    ds.w    1     ;Flags (Aussehen des Gadgets)
10          gg_GadgetType   ds.w    1     ;Gadget-Typ (Bool/Str/Prop)
12          gg_GadgetRender ds.l    1     ;^Image oder ^Border
16          gg_SelectRender ds.l    1     ;^Image wenn angeklickt
1A          gg_GadgetText   ds.l    1     ;^Text-Struktur
1E          gg_MutualExclude ds.l   1     ; Gadget abschalten
22          gg_SpecialInfo  ds.l    1     ;^String- oder Prop-Info
26          gg_GadgetID     ds.w    1     ;beliebige User-ID
28          gg_UserData     ds.l    1     ;^beliebige Daten
2C          gg_SIZEOF       equ     $2C   ;Größe der Struktur

AUTOFRONTPEN    equ   0              ;empfohlene Werte für Auto-Request
AUTOBACKPEN     equ   1              ;siehe IntuitionText
AUTODRAWMODE    equ   1
AUTOLEFTEDGE    equ   6
AUTOTOPEDGE     equ   3
AUTOITEXTFONT   equ   0
AUTONEXTTEXT    equ   0
```

```
GADGHCOMP       equ $0000    ;bei Anklicken Invertieren
GADGHBOX        equ $0001    ;bei Anklicken Box um Gadget
GADGHIMAGE      equ $0002    ;bei Anklicken alternatives Image/Border
GADGHNONE       equ $0003    ;bei Anklicken keine Reaktion

GADGIMAGE       equ $0004    ;0/1=Border/Image
GRELBOTTOM      equ $0008    ;0/1= relativ zu Top/Bottom-Grenze
GRELRIGHT       equ $0010    ;0/1= relativ zu links/rechts
GRELWIDTH       equ $0020    ;absolute/relative Breite
GRELHEIGHT      equ $0040    ;                  Höhe
SELECTED        equ $0080    ;Vorwahl aus/an
GADGDISABLED    equ $0100    ;für An/Aus-Gadgets
RELVERIFY       equ $0001    ;Release Verify
GADGIMMEDIATE   equ $0002    ;sofort Nachricht wenn
ENDGADGET       equ $0004    ;Requester vom Schirm
FOLLOWMOUSE     equ $0008    ;sende Maus-Koordinaten
RIGHTBORDER     equ $0010    ;justieren nach rechts
LEFTBORDER      equ $0020    ;               links
TOPBORDER       equ $0040    ;               oben
BOTTOMBORDER    equ $0080    ;               unten
TOGGLESELECT    equ $0100    ;toggelt Gadget
STRINGCENTER    equ $0200    ;Text zentrieren
STRINGRIGHT     equ $0400    ;Text rechtsbündig
LONGINT         equ $0800    ;erlaube Long Int im String Gadget
ALTKEYMAP       equ $1000    ;wenn vorhanden und in StringInfo

BOOLGADGET      equ $0001
GADGET0002      equ $0002
PROPGADGET      equ $0003
STRGADGET       equ $0004

GADGETTYPE      equ $FC00
SYSGADGET       equ $8000    ;System-Gadget (vergibt Intuition)
SCRGADGET       equ $4000    ;wenn im Screen
GZZGADGET       equ $2000    ;wenn im Gimmezerozero-Window
REQGADGET       equ $1000    ;wenn im Requester
SIZING          equ $0010    ;Sys-Typen
WDRAGGING       equ $0020
SDRAGGING       equ $0030
WUPFRONT        equ $0040
SUPFRONT        equ $0050
WDOWNBACK       equ $0060
SDOWNBACK       equ $0070
CLOSE           equ $0080
```

```
;PropInfo-Struktur (für Proportional-Gadgets)

00          pi_Flags        ds.w    1   ;Flags
02          pi_HorizPot     ds.w    1   ;horizontale Schieberposition
04          pi_VertPot      ds.w    1   ;vertikale Schieberposition
06          pi_HorizBody    ds.w    1   ;horizontale Schrittweite
08          pi_VertBody     ds.w    1   ;vertikale Schrittweite
0A          pi_CWidth       ds.w    1   ;Breite, Rahmen
0C          pi_CHeight      ds.w    1   ;Höhe
0E          pi_HPotRes      ds.w    1   ;absolute Schrittweite horizontal
10          pi_VPotRes      ds.w    1   ;absolute Schrittweite vertikal
12          pi_LeftBorder   ds.w    1   ;Rahmenlage links
14          pi_TopBorder    ds.w    1   ;Rahmenlage oben
16          pi_SIZEOF       equ     $16 ;Länge der Struktur

AUTOKNOB        equ $0001    ;automatischer Knopf
FREEHORIZ       equ $0002    ;Knopfbewegung horizontal
FREEVERT        equ $0004    ;              vertikal
PROPBORDERLESS  equ $0008    ;ohne Border
KNOBHIT         equ $0100    ;1, wenn Knopf berührt wird
KNOBHMIN        equ 6
KNOBVMIN        equ 4
MAXBODY         equ $FFFF
MAXPOT          equ $FFFF

;StringInfo-Struktur (für String-Gadgets)

00          si_Buffer       ds.l    1       ;^Textpuffer
04          si_UndoBuffer   ds.l    1       ;^Undo-Buffer
08          si_BufferPos    ds.w    1       ;Anfangs-Cursor-Position
0A          si_MaxChars     ds.w    1       ;Puffergröße + 1;
0C          si_DispPos      ds.w    1       ;Pos. Cursor-Zeichen
0E          si_UndoPos      ds.w    1       ;Cursor in Undo-Buffer
10          si_NumChars     ds.w    1       ;Zeichen im Puffer
12          si_DispCount    ds.w    1       ;Zeichen sichtbar
14          si_CLeft        ds.w    1       ;horizontaler Container-Offset
16          si_CTop         ds.w    1       ;vertikaler Container-Offset
18          si_LayerPtr     ds.l    1       ;^aktueller Rastport
1C          si_LongInt      ds.l    1       ;Long Int hier
20          si_AltKeyMap    ds.l    1       ;^eigene Keymap
            si_SIZEOF       equ     $24     ;Strukturlänge in Bytes

;Intuition-Text-Struktur

00          it_FrontPen     ds.b    1       ;Vordergrund-Farbe
01          it_BackPen      ds.b    1       ;Hintergrund-Farbe
02          it_DrawMode     ds.b    1  ;Textdarstellung JAM1, JAM2 oder XOR
```

```
03          it_AdjustToWord ds.b    1
04          it_LeftEdge     ds.w    1  ;linke Ecke
06          it_TopEdge      ds.w    1  ;obere Ecke
08          it_ITextFont    ds.l    1  ;^Font-Struktur oder 0
0C          it_IText        ds.l    1  ;^Text-String (0-term.)
10          it_NextText     ds.l    1  ;^Nächste Struktur oder 0
            it_SIZEOF       equ     $14;Strukturlänge in Bytes

;Border-Struktur

00          bd_LeftEdge     ds.w    1  ;linke Ecke
02          bd_TopEdge      ds.w    1  ;obere Ecke
04          bd_FrontPen     ds.b    1  ;Vordergrund-Farbe
05          bd_BackPen      ds.b    1  ;Hintergrund (ohne Wirkung)
06          bd_DrawMode     ds.b    1  ;Darstellung JAM1 oder XOR
07          bd_Count        ds.b    1  ;Anzahl Koordinatenpaare
08          bd_XY           ds.l    1  ;^Array mit Koordinatenpaaren
0C          bd_NextBorder   ds.l    1  ;^nächstes oder 0
10          bd_SIZEOF       equ     $10;Strukturlänge in Bytes

;Image-Struktur

00          ig_LeftEdge     ds.w    1  ;linke Ecke
02          ig_TopEdge      ds.w    1  ;obere Ecke
04          ig_Width        ds.w    1  ;Breite
06          ig_Height       ds.w    1  ;Höhe
08          ig_Depth        ds.w    1  ;Anzahl Bitplanes
0A          ig_ImageData    ds.l    1  ;^Bitmuster
0E          ig_PlanePick    ds.b    1  ;genutzte Planes
0F          ig_PlaneOnOff   ds.b    1  ;Hintergrund
10          ig_NextImage    ds.l    1  ;^nächste Struktur
14          ig_SIZEOF       equ     $14;Strukturlänge in Bytes

;Intuition-Message-Struktur

00-13       im_ExecMessage  ds.b    MN_SIZE ;Exec-Message-Struktur
14          im_Class        ds.l    1       ;Bits wie IDCMP-Flags
18          im_Code         ds.w    1       ;Werte hier
1A          im_Qualifier    ds.w    1       ;für RAW-IO
1C          im_IAddress     ds.l    1       ;Adresse von Objekten
20          im_MouseX       ds.w    1       ;Mouse-Koordinate X
22          im_MouseY       ds.w    1       ;Mouse-Koordinate Y
24          im_Seconds      ds.l    1       ;System-Zeit
28          im_Micros       ds.l    1       ;
2C          im_IDCMPWindow  ds.l    1       ;Adresse des Fensters
30          im_SpecialLink  ds.l    1       ;reserviert
            im_SIZEOF       equ     $34     ;Strukturlänge in Bytes
```

```
SIZEVERIFY      equ   $00000001    ;Message, wenn Versuch Sizing
NEWSIZE         equ   $00000002    ;Message, wenn Sizing fertig
REFRESHWINDOW   equ   $00000004    ;Message, wenn Refresh nötig
MOUSEBUTTONS    equ   $00000008    ;Message, wenn Mouse-Events
MOUSEMOVE       equ   $00000010    ;
GADGETDOWN      equ   $00000020    ;Message, wenn Gadget-Event
GADGETUP        equ   $00000040    ;
REQSET          equ   $00000080    ;Message, wenn Requester
MENUPICK        equ   $00000100    ;Message, wenn Menu-Event
CLOSEWINDOW     equ   $00000200    ;Message, wenn Close_Gadget
RAWKEY          equ   $00000400    ;Message, wenn Raw-Key
REQVERIFY       equ   $00000800    ;Warte bevor Requester erlaubt
REQCLEAR        equ   $00001000    ;Message, wenn letzer Requ. weg
MENUVERIFY      equ   $00002000    ;Warte bis Menus gezeichnet
NEWPREFS        equ   $00004000    ;Message, wenn Prefs. geändert
DISKINSERTED    equ   $00008000    ;Message,  wenn   Diskette
DISKREMOVED     equ   $00010000    ;                 rein/raus
WBENCHMESSAGE   equ   $00020000    ;
ACTIVEWINDOW    equ   $00040000    ;
INACTIVEWINDOW  equ   $00080000
DELTAMOVE       equ   $00100000    ;Mouse-Pos. relativ
VANILLAKEY      equ   $00200000    ;Message, wenn Key in Code
INTUITICKS      equ   $00400000    ;Message nach 1/10Sekunde
LONELYMESSAGE   equ   $80000000
;für Menu-Verify:
MENUHOT         equ   $000     1   ;Cancel muß verifiziert werden
MENUCANCEL      equ   $0002        ;Hot Reply cancelt Menu
MENUWAITING     equ   $0003        ;Int. wartet auf Reply

WBENCHOPEN      equ $0001
WBENCHCLOSE     equ $0002

;NewWindow-Struktur

00             nw_LeftEdge     ds.w    1      ;linke Ecke
02             nw_TopEdge      ds.w    1      ;obere Ecke
04             nw_Width        ds.w    1      ;Breite
06             nw_Height       ds.w    1      ;Höhe
08             nw_DetailPen    ds.b    1      ;Schriftfarbe
09             nw_BlockPen     ds.b    1      ;Hintergrundfarbe
0A             nw_IDCMPFlags   ds.l    1      ;Bits siehe unten
0E             nw_Flags        ds.l    1      ;Fenstertyp
12             nw_FirstGadget  ds.l    1      ;^User-Gadgets
16             nw_CheckMark    ds.l    1      ;^User-Menühaken
1A             nw_Title        ds.l    1      ;^Titel-Text
1E             nw_Screen       ds.l    1      ;^Screen-Struktur
22             nw_BitMap       ds.l    1      ;^User-Bitmap
```

```
26              nw_MinWidth      ds.w    1       ;Min. Breite
28              nw_MinHeight     ds.w    1       ;     Höhe
2A              nw_MaxWidth      ds.w    1       ;max. Breite
2C              nw_MaxHeight     ds.w    1       ;     Höhe
2E              nw_Type          ds.w    1       ;Screen-Typ
                nw_SIZE          equ     $30     ;Strukturgröße in Bytes

WINDOWSIZING     equ $0001      ;erlaubte Gadgets
WINDOWDRAG       equ $0002
WINDOWDEPTH      equ $0004
WINDOWCLOSE      equ $0008

SIZEBRIGHT       equ $0010      ;Sizing-Gadget rechts außen
SIZEBBOTTOM      equ $0020      ;                     innen

REFRESHBITS      equ $00C0      ;eines sollte sein:
SMART_REFRESH    equ $0000      ;
SIMPLE_REFRESH   equ $0040      ;
SUPER_BITMAP     equ $0080      ;
OTHER_REFRESH    equ $00C0      ;

BACKDROP         equ $0100      ;wenn Backdrop-Window
REPORTMOUSE      equ $0200      ;Message wenn Mouse-Event
GIMMEZEROZERO    equ $0400      ;wenn das gewünscht
BORDERLESS       equ $0800      ;wenn kein Rahmen

ACTIVATE         equ $1000      ;aktiv nach Open
WINDOWACTIVE     equ $2000      ;Message wenn aktiviert
INREQUEST        equ $4000      ;Wd in Request-Modus
MENUSTATE        equ $8000      ;Message, wenn Menu

RMBTRAP          equ $00010000 ;Message, wenn rechte Maustaste
NOCAREREFRESH    equ $00020000 ;keine Message bei Refresh

WINDOWREFRESH    equ $01000000
WBENCHWINDOW     equ $02000000
WINDOWTICKED     equ $04000000

SUPER_UNUSED     equ $FCFC0000

;Window-Struktur (wird von Intuition angelegt)

00              wd_NextWindow    ds.l    1       ;^nächstes Window
04              wd_LeftEdge      ds.w    1       ;linke Ecke
06              wd_TopEdge       ds.w    1       ;obere Ecke
08              wd_Width         ds.w    1       ;Breite
0A              wd_Height        ds.w    1       ;Höhe
```

```
0C              wd_MouseX         ds.w    1      ;Mouse-Koordinate X
0E              wd_MouseY         ds.w    1      ;Mouse-Koordiante Y
10              wd_MinWidth       ds.w    1      ;min. Breite
12              wd_MinHeight      ds.w    1      ;     Höhe
14              wd_MaxWidth       ds.w    1      ;max. Breite
16              wd_MaxHeight      ds.w    1      ;     Höhe
18              wd_Flags          ds.l    1      ;Window-Flags siehe oben
1C              wd_MenuStrip      ds.l    1      ;^Menu-Liste
20              wd_Title          ds.l    1      ;^Titel-Text
24              wd_FirstRequest   ds.l    1      ;^erster Requester
28              wd_DMRequest      ds.l    1      ;^DM-Requester
2C              wd_ReqCount       ds.w    1      ;Anzahl Requester
2E              wd_WScreen        ds.l    1      ;^Screen-Struktur
32              wd_RPort          ds.l    1      ;^RastPort-Struktur
36              wd_BorderLeft     ds.b    1      ;aktuelle Lage von
37              wd_BorderTop      ds.b    1      ;linker,oberer und
38              wd_BorderRight    ds.b    1      ;rechter,unterer
39              wd_BorderBottom   ds.b    1      ;Ecke
3A              wd_BorderRPort    ds.l    1      ;^RastPort
3E              wd_FirstGadget    ds.l    1      ;^Gadget-Liste
42              wd_Parent         ds.l    1      ;in Liste
46              wd_Descendant     ds.l    1      ;
4A              wd_Pointer        ds.l    1      ;^Mauszeiger-Struktur
4E              wd_PtrHeight      ds.b    1      ;            Höhe
4F              wd_PtrWidth       ds.b    1      ;            Breite
50              wd_XOffset        ds.b    1      ;Hot-Point Offset X
51              wd_YOffset        ds.b    1      ;Hot-Point Offset Y
52              wd_IDCMPFlags     ds.l    1      ;Bits siehe oben
56              wd_UserPort       ds.l    1      ;^Empfangsport-Struktur
5A              wd_WindowPort     ds.l    1      ;^Sende-Port-Struktur
5E              wd_MessageKey     ds.l    1      ;^Int.-Message
62              wd_DetailPen      ds.b    1      ;Zeichenfarbe
63              wd_BlockPen       ds.b    1      ;Hintergrundfarbe
64              wd_CheckMark      ds.l    1      ;^User-Menühaken
68              wd_ScreenTitle    ds.l    1      ;^Screen-Titel (0)
6C              wd_GZZMouseX      ds.w    1      ;Größe und Mauskoordinaten
6E              wd_GZZMouseY      ds.w    1      ;bei Verwendung eines
70              wd_GZZWidth       ds.w    1      ;Gimmezerozero-Windows
72              wd_GZZHeight      ds.w    1
74              wd_ExtData        ds.l    1      ;zwei Zeiger für
78              wd_UserData       ds.l    1      ;User
7C              wd_WLayer         ds.l    1      ;^Layer-Struktur des Windows
```

```
;NewScreen-Struktur

00              ns_LeftEdge     ds.w    1      ;linke Ecke
02              ns_TopEdge      ds.w    1      ;obere Ecke
04              ns_Width        ds.w    1      ;Breite
06              ns_Height       ds.w    1      ;Höhe
08              ns_Depth        ds.w    1      ;Anzahl Bitplanes
0A              ns_DetailPen    ds.b    1      ;Zeichenfarbe
0B              ns_BlockPen     ds.b    1      ;Hintergrundfarbe
0C              ns_ViewModes    ds.w    1      ;Screen-Modus
0E              ns_Type         ds.w    1      ;CUSTOM oder Bitmap
10              ns_Font         ds.l    1      ;^Zeichensatz oder 0
14              ns_DefaultTitle ds.l    1      ;^Titeltext
18              ns_Gadgets      ds.l    1      ;Immer 0 setzen!!
1C              ns_CustomBitMap ds.l    1      ;^auf eigene Bitmap
                ns_SIZEOF       equ     $20    ;Strukturgröße in Bytes


SCREENTYPE      equu   $000F   ;alle Typen
WBENCHSCREEN    equ    $0001   ;Workbench
CUSTOMSCREEN    equ    $000F   ;eigener
SHOWTITLE       equ    $0010   ;1 wenn ShowTitle gerufen
BEEPING         equ    $0020   ;1 wenn Beep (Blink)
CUSTOMBITMAP    equ    $0040   ;1 wenn eigene Bitmap

FILENAME_SIZE   equ 30

POINTERSIZE     equ    36

TOPAZ_EIGHTY    equ    8
TOPAZ_SIXTY     equ    9


;Screen-Struktur (wird von Intuition angelegt)

000             sc_NextScreen   ds.l    1       ;^nächste Screenstruktur
004             sc_FirstWindow  ds.l    1       ;^Struktur erstes Window
008             sc_LeftEdge     ds.w    1       ;linke Ecke
00A             sc_TopEdge      ds.w    1       ;obere Ecke
00C             sc_Width        ds.w    1       ;Breite
00E             sc_Height       ds.w    1       ;Höhe
010             sc_MouseY       ds.w    1       ;Maus-Koordinate Y
012             sc_MouseX       ds.w    1       ;Maus-Koordinate X
014             sc_Flags        ds.w    1       ;Bits siehe oben
016             sc_Title        ds.l    1       ;^Titel-Text
01A             sc_DefaultTitle ds.l    1       ;^Text für Windows ohne Screen
01E             sc_BarHeight    ds.b    1       ;Dimensionen der Titelleiste
```

```
01F          sc_BarVBorder   ds.b   1         ;  des Screens,
020          sc_BarHBorder   ds.b   1         ;
021          sc_MenuVBorder  ds.b   1         ;  der Menüs,
022          sc_MenuHBorder  ds.b   1
023          sc_WBorTop      ds.b   1         ;  und aller Windows
024          sc_WBorLeft     ds.b   1         ;  dieses Screens
025          sc_WBorRight    ds.b   1
026          sc_WBorBottom   ds.b   1
027          sc_AdjustToWord ds.b   1
028          sc_Font         ds.l   1         ;^Default Font
02C-053      sc_ViewPort     ds.b   vp_SIZEOF;View-Port-Struktur
054-0B7      sc_RastPort     ds.b   rp_SIZEOF;Rast-Port-Struktur
0B8-0DF      sc_BitMap       ds.b   $28       ;ext. Bitmap-Struktur
0E0          sc_LayerInfo    ds.b   li_SIZEOF;Layer-Info-Struktur
146          sc_FirstGadget  ds.l   1         ;^Gadget-Liste
14A          sc_DetailPen    ds.b   1         ;Zeichenfarbe
14B          sc_BlockPen     ds.b   1         ;Hintergrundfarbe
14C          sc_SaveColor0   ds.w   1         ;für Beep
14E          BarLayer        ds.l   1         ;sc_ fehlt wirklich!
152          sc_ExtData      ds.l   1         ;^User-Data
156          sc_UserData     ds.l   1
             sc_SIZEOF       equ    $15A      ;Strukturgröße in Bytes


;Preferences-Struktur

00         pf_FontHeight     ds.b 1             ;Zeichensatzgröße
01         pf_PrinterPort    ds.b 1             ;Druckerport
02         pf_BaudRate       ds.w 1             ;Modem-Übertragungsrate
04-0B      pf_KeyRptSpeed    ds.b TV_SIZE       ;Strukturen für
0C-13      pf_KeyRptDelay    ds.b TV_SIZE       ;Tastenwiederholung,
14-1B      pf_DoubleClick    ds.b TV_SIZE       ;Doppel-Klick
1C-63      pf_PointerMatrix  ds.b POINTERSIZE*2 ;Maus-Zeiger-Grafikdaten
64         pf_XOffset        ds.b 1             ;Hot-Point-Offset X
65         pf_YOffset        ds.b 1             ;Hot-Point-Offset Y
66         pf_color17        ds.w 1             ;Farben-Mauszeiger
68         pf_color18        ds.w 1
6A         pf_color19        ds.w 1
6C         pf_PointerTicks   ds.w 1             ;Mauszeiger-Speed
6E         pf_color0         ds.w 1             ;Voreinstellung
70         pf_color1         ds.w 1             ;Bildschirmfarben
72         pf_color2         ds.w 1
74         pf_color3         ds.w 1
76         pf_ViewXOffset    ds.b 1             ;Daten zur Bildschirmlage
77         pf_ViewYOffset    ds.b 1
78         pf_ViewInitX      ds.w 1
7A         pf_ViewInitY      ds.w 1
```

```
7C           EnableCLI          ds.w  1
7E           pf_PrinterType     ds.w  1                 ;Druckertyp
80-9D        pf_PrinterFilename  ds.b  FILENAME_SIZE ;Filename
                                                         Druckertreiber
9E             pf_PrintPitch       ds.w  1              ;Daten für Drucker
A0             pf_PrintQuality     ds.w  1              ;(Draft, NLQ,
A2             pf_PrintSpacing     ds.w  1              ;Zeilenabstand, etc.)
A4             pf_PrintLeftMargin  ds.w  1
A6             pf_PrintRightMargin ds.w  1
A8             pf_PrintImage       ds.w  1
AA             pf_PrintAspect      ds.w  1
AC             pf_PrintShade       ds.w  1
AE             pf_PrintThreshold   ds.w  1
B0             pf_PaperSize        ds.w  1              ;Daten für Papier
B2             pf_PaperLength      ds.w  1              ;(Länge, Endlos,
B4             pf_PaperType        ds.w  1              ;Einzelblatt)
B6             pf_padding          ds.b  50
               pf_SIZEOF           equ   $E8            ;Strukturlänge in Bytes


PARALLEL_PRINTER equ   $00
SERIAL_PRINTER   equ   $01
BAUD_110         equ   $00
BAUD_300         equ   $01
BAUD_1200        equ   $02
BAUD_2400        equ   $03
BAUD_4800        equ   $04
BAUD_9600        equ   $05
BAUD_19200       equ   $06
BAUD_MIDI        equ   $07

FANFOLD          equ   $00
SINGLE           equ   $80
PICA             equ   $000
ELITE            equ   $400
FINE             equ   $800
DRAFT            equ   $000
LETTER           equ   $100

SIX_LPI          equ   $000
EIGHT_LPI        equ   $200

IMAGE_POSITIVE   equ   0
IMAGE_NEGATIVE   equ   1

ASPECT_HORIZ     equ   0
ASPECT_VERT      equ   1
```

```
SHADE_BW          equ   $00
SHADE_GREYSCALE   equ   $01
SHADE_COLOR       equ   $02

US_LETTER         equ   $00
US_LEGAL          equ   $10
N_TRACTOR         equ   $20
W_TRACTOR         equ   $30
CUSTOM            equ   $40
CUSTOM_NAME       equ   $00

ALPHA_P_101       equ   $01
BROTHER_15XL      equ   $02
CBM_MPS1000       equ   $03
DIAB_630          equ   $04
DIAB_ADV_D25      equ   $05
DIAB_C_150        equ   $06
EPSON             equ   $07
EPSON_JX_80       equ   $08
OKIMATE_20        equ   $09
QUME_LP_20        equ   $0A
HP_LASERJET       equ   $0B
HP_LASERJET_PLUS  equ   $0C


;Remember-Struktur

00          rm_NextRemember ds.l   1     ;^nächster Knoten
04          rm_RememberSize ds.l   1     ;Größe
08          rm_Memory       ds.l   1     ;^Adresse
0C          rm_SIZEOF       ds.w   0


;Alert-Typen

ALERT_TYPE       equ   $80000000
RECOVERY_ALERT   equ   $00000000
DEADEND_ALERT    equ   $80000000
```

```
SHADE_GREYSCALE  equ   $01
SHADE_COLOR      equ   $02

US_LETTER        equ   $00
US_LEGAL         equ   $10
N_TRACTOR        equ   $20
W_TRACTOR        equ   $30
CUSTOM           equ   $40
CUSTOM_NAME      equ   $00

ALPHA_P_101      equ   $01
BROTHER_15XL     equ   $02
CBM_MPS1000      equ   $03
DIAB_630         equ   $04
DIAB_ADV_D25     equ   $05
DIAB_C_150       equ   $06
EPSON            equ   $07
EPSON_JX_80      equ   $08
OKIMATE_20       equ   $09
QUME_LP_20       equ   $0A
HP_LASERJET      equ   $0B
HP_LASERJET_PLUS equ   $0C


;Remember-Struktur

00             rm_NextRemember ds.l    1      ;^nächster Knoten
04             rm_RememberSize ds.l    1      ;Größe
08             rm_Memory       ds.l    1      ;^Adresse
0C             rm_SIZEOF       ds.w    0


;Alert-Typen

ALERT_TYPE      equ   $80000000
RECOVERY_ALERT  equ   $00000000
DEADEND_ALERT   equ   $80000000
```

4. Graphics

```
;zu Testzwecken importiert vom:

MP_SIZE equ     $22     ;ports
LH_SIZE equ     $0E     ;lists
MN_SIZE equ     $14     ;ports
IS_SIZE equ     $16     ;libraries
```

```
LIB_SIZE equ    $22


;Layer-Structure

00   lr_Front          ds.l   1        ;^Layer über diesem
04   lr_Back           ds.l   1        ;^Layer unter diesem
08   lr_ClipRect       ds.l   1        ;^Clipping-Rechteck-Struktur
0C   lr_RastPort       ds.l   1        ;^Rastport
10   lr_MinX           ds.w   1        ;Clipping-Rechteck:
12   lr_MinY           ds.w   1
14   lr_MaxX           ds.w   1
16   lr_MaxY           ds.w   1
18   lr_Lock           ds.b   1        ;Task-Lock des Layers
19   lr_LockCount      ds.b   1        ;Anzahl Tasks zu
1A   lr_LayerLockCount ds.b   1        ;  diesem Layer
1B   lr_reserved       ds.b   1
1C   lr_reserved1      ds.w   1
1E   lr_Flags          ds.w   1        ;16 Bit = Typ
20   lr_SuperBitMap    ds.l   1        ;^Super-Bitmap
24   lr_SuperClipRect  ds.l   1        ;^ClipRect wenn S-Bitmap-Lay.
28   lr_Window         ds.l   1        ;^Intuition-Window
2C   lr_Scroll_ X      ds.w   1        ;Scro-Weite in Pixels
2E   lr_Scroll_Y       ds.w   1
30   lr_LockPort       ds.b   MP_SIZE ;Name Msg-Port
52   lr_LockMessage    ds.b   MN_SIZE ;Msg-Struktur
66   lr_ReplyPort      ds.b   MP_SIZE ;Name Msg-Port
88   lr_1_LockMessage  ds.b   MN_SIZE ;Msg-Struktur
9C   lr_DamageList     ds.l   1        ;^Region-Struktur
A0   lr__cliprects     ds.l   1        ;^Clip-Rect
A4   lr_LayerInfo      ds.l   1        ;^Layerinfo-Stuktur
A8   lr_LayerLocker    ds.l   1        ;^Task-Struktur
AC   lr_SuperSaverClipRects  ds.l    1 ;System-Use:
B0   lr_cr             ds.l   1
B4   lr_cr2            ds.l   1
B8   lr_crnew          ds.l   1
BC   lr__p1            ds.l   1
C0   lr_SIZEOF         ds.w   0

;Clip-Rect
00   cr_Next          ds.l   1         ;^Nachfolger
04   cr_Prev          ds.l   1         ;^Vorgänger
08   cr_LObs          ds.l   1         ;System-Use
0C   cr_BitMap        ds.l   1         ;^Super-Bitmap
10   cr_MinX          ds.w   1         ;Rechteck:
12   cr_MinY          ds.w   1
14   cr_MaxX          ds.w   1
```

```
16    cr_MaxY          ds.w    1
18    cr__p1           ds.l    1           ;System-Use:
1C    cr__p2           ds.l    1
20    cr_reserved      ds.l    1
24    cr_Flags         ds.l    1
28    cr_SIZEOF        ds.w    0


ISLESSX equ 1
ISLESSY equ 2
ISGRTRX equ 4
ISGRTRY equ 8

;copper

COPPER_MOVE     equ     0                  ;Pseudo-Op-Codes
COPPER_WAIT     equ     1
CPRNXTBUF       equ     2
CPR_NT_LOF      equ     $8000
CPR_NT_SHT      equ     $4000

00    ci_OpCode        ds.w    1           ;Op-Codes
02    ci_nxtlist       ds.b    0
02    ci_VWaitPos      ds.b    0
02    ci_DestAddr      ds.b    2
04    ci_HWaitPos      ds.b    0
04    ci_DestData      ds.b    2
06    ci_SIZEOF        ds.w    0

00    crl_Next         ds.l    1           ;^aktuelle Copper-Liste
04    crl_start        ds.l    1
08    crl_MaxCount     ds.w    1
0A    crl_SIZEOF       ds.w    0

;Copper-Liste
00    cl_Next          ds.l    1           ;^Nachfolger
04    cl__CopList      ds.l    1           ;System-Use
08    cl_ViewPort      ds.l    1           ;
0C    cl_CopIns        ds.l    1           ;
10    cl_CopPtr        ds.l    1
14    cl_CopLStart     ds.l    1
18    cl_CopSStart     ds.l    1
1C    cl_Count         ds.w    1
1E    cl_MaxCount      ds.w    1
20    cl_DyOffset      ds.w    1
22    cl_SIZEOF        ds.w    0
```

```
00   ucl_Next           ds.l    1      ;Cop-List-Header
04   ucl_FirstCopList   ds.l    1
08   ucl_CopList        ds.l    1
0C   ucl_SIZEOF         ds.w    0

00   copinit_diagstrt   ds.b    8      ;interne Cop-Struktur
08   copinit_sprstrtup  ds.b    80
58   copinit_sprstop    ds.b    4
5C   copinit_SIZEOF     ds.w    0

;Gels (Grafik-Elemente)

SUSERFLAGS       equu    $0F
VSB_VSPRITE      equ     0
VSF_VSPRITE      equ     1
VSB_SAVEBACK     equ     1
VSF_SAVEBACK     equ     2
VSB_OVERLAY      equ     2
VSF_OVERLAY      equ     4
VSB_MUSTDRAW     equ     3
VSF_MUSTDRAW     equ     8
VSB_BACKSAVED    equ     8
VSF_BACKSAVED    equ     $100
VSB_BOBUPDATE    equ     9
VSF_BOBUPDATE    equ     $200
VSB_GELGONE      equ     10
VSF_GELGONE      equ     $400
VSB_VSOVERFLOW   equ     11
VSF_VSOVERFLOW   equ     $800


BUSERFLAGS       equ     $0FF
BB_SAVEBOB       equ     0
BF_SAVEBOB       equ     1
BB_BOBISCOMP     equ     1
BF_BOBISCOMP     equ     2
BB_BWAITING      equ     8
BF_BWAITING      equ     $100
BB_BDRAWN        equ     9
BF_BDRAWN        equ     $200
BB_BOBSAWAY      equ     10
BF_BOBSAWAY      equ     $400
BB_BOBNIX        equ     11
BF_BOBNIX        equ     $800
BB_SAVEPRESERVE  equ     12
BF_SAVEPRESERVE  equ     $1000
BB_OUTSTEP       equ     13
```

```
BF_OUTSTEP      equ     $2000

ANFRACSIZE      equ     6
ANIMHALF        equ     $20
RINGTRIGGER     equ     1

;V-Sprite-Struktur (auch fuer Bobs)
00   vs_NextVSprite ds.l    1         ;^Nachfolger
04   vs_PrevVSprite ds.l    1         ;^Vorgänger
08   vs_DrawPath    ds.l    1         ;System-Use
0C   vs_ClearPath   ds.l    1         ;System-Use
10   vs_Oldy        ds.w    1         ;Vorherige Pos. Y
12   vs_Oldx        ds.w    1         ;               X
14   vs_VSFlags     ds.w    1         ;ist:
16   vs_Y           ds.w    1
18   vs_X           ds.w    1
1A   vs_Height      ds.w    1         ;Höhe des Sprite
1C   vs_Width       ds.w    1         ;Breite
1E   vs_Depth       ds.w    1         ;Bit-Planes
20   vs_MeMask      ds.w    1         ;Masken für
22   vs_HitMask     ds.w    1         ;Kollisions-Handling
24   vs_ImageData   ds.l    1         ;^Daten
28   vs_BorderLine  ds.l    1         ;^Puffer
2C   vs_CollMask    ds.l    1         ;^Kollisions-Maske
30   vs_SprColors   ds.l    1         ;^Color-Tabelle
34   vs_VSBob       ds.l    1         ;^Bob, wenn Bob
38   vs_PlanePick   ds.b    1         ;Plane-Maske, wenn Bob
39   vs_PlaneOnOff  ds.b    1         ;
3A   vs_SUserExt    ds.w    0         ;evtl. User-Extensions
3A   vs_SIZEOF      ds.w    0

;Bobs
00   bob_BobFlags    ds.w   1         ;Aspekt-Bits
02   bob_SaveBuffer  ds.l   1         ;^Puffer
06   bob_ImageShadow ds.l   1         ;^Shadowmask
0A   bob_Before      ds.l   1         ;^Vorgänger
0E   bob_After       ds.l   1         ;^Nachfolger
12   bob_BobVSprite  ds.l   1         ;
16   bob_BobComp     ds.l   1
1A   bob_DBuffer     ds.l   1
1E   bob_BUserExt    ds.w   0
1E   bob_SIZEOF      ds.w   0

;Animations-Ablauf
00   ac_CompFlags    ds.w   1         ;Typ-Bits
02   ac_Timer        ds.w   1         ;Ist-Zeit
04   ac_TimeSet      ds.w   1         ;Vorgabe
```

```
06    ac_NextComp      ds.l   1         ;^Nachfolger
0A    ac_PrevComp      ds.l   1         ;^Vorgänger
0E    ac_NextSeq       ds.l   1         ;dto in
12    ac_PrevSeq       ds.l   1         ;Zeichensequenz
16    ac_AnimCRoutine  ds.l   1         ;^Exit-Routine (0)
1A    ac_YTrans        ds.w   1         ;Anfangsdistanz
1C    ac_XTrans        ds.w   1
1E    ac_HeadOb        ds.l   1         ;^AminOb-Structure
22    ac_AnimBob       ds.l   1         ;^Bob-Structure
26    ac_SIZE          ds.w   0

;Animations-Objekt

00    ao_NextOb        ds.l   1         ;^Nachfolger
04    ao_PrevOb        ds.l   1         ;^Vorgänger
08    ao_Clock         ds.l   1         ;Aufrufe
0C    ao_AnOldY        ds.w   1         ;Alte Lage
0E    ao_AnOldX        ds.w   1
10    ao_AnY           ds.w   1         ;ist
12    ao_AnX           ds.w   1
14    ao_YVel          ds.w   1         ;Speed
16    ao_XVel          ds.w   1
18    ao_XAccel        ds.w   1         ;Beschleunigung
1A    ao_YAccel        ds.w   1
1C    ao_RingYTrans    ds.w   1         ;Inkrements
1E    ao_RingXTrans    ds.w   1
20    ao_AnimORoutine  ds.l   1         ;^Routine
24    ao_HeadComp      ds.l   1         ;^erstes Objekt
28    ao_AUserExt      ds.w   0
28    ao_SIZEOF        ds.w   0

00    dbp_BufY         ds.w   1         ;Zwischenpuffer
02    dbp_BufX         ds.w   1
04    dbp_BufPath      ds.l   1
08    dbp_BufBuffer    ds.l   1
0C    dbp_BufPlanes    ds.l   1
10    dbp_SIZEOF       ds.w   0


;gfxbase

22    gb_ActiView      ds.l   1
26    gb_copinit       ds.l   1
2A    gb_cia           ds.l   1
2E    gb_blitter       ds.l   1
32    gb_LOFlist       ds.l   1
36    gb_SHFlist       ds.l   1
```

```
3A    gb_blthd            ds.l    1
3E    gb_blttl            ds.l    1
42    gb_bsblthd          ds.l    1
46    gb_bsblttl          ds.l    1
4A    gb_vbsrv            ds.b    IS_SIZE
60    gb_timsrv           ds.b    IS_SIZE
76    gb_bltsrv           ds.b    IS_SIZE
8C    gb_TextFonts        ds.b    LH_SIZE
9A    gb_DefaultFont      ds.l    1
9E    gb_Modes            ds.w    1
A0    gb_VBlank           ds.b    1
A1    gb_Debug            ds.b    1
A2    gb_BeamSync         ds.w    1
A4    gb_system_bplcon0   ds.w    1
A6    gb_SpriteReserved   ds.b    1
A7    gb_bytereserved     ds.b    1
A8    gb_Flags            ds.w    1
AA    gb_BlitLock         ds.w    1
AC    gb_BlitNest         ds.w    1
AE    gb_BlitWaitQ        ds.b    LH_SIZE
BC    gb_BlitOwner        ds.l    1
C0    gb_TOF_WaitQ        ds.b    LH_SIZE
CE    gb_DisplayFlags     ds.w    1
D0    gb_SimpleSprites    ds.l    1
D4    gb_MaxDisplayRow    ds.w    1
D6    gb_reserved         ds.b    8
DE    gb_SIZE             ds.w    0


OWNBLITTERn     equ    0
QBOWNERn        equ    1
QBOWNER         equ    2

;GFX

BITSET  equ $8000
BITCLR  equ 0
AGNUS   equ 1
DENISE  equ 1

00    bm_BytesPerRow    ds.w    1
02    bm_Rows           ds.w    1
04    bm_Flags          ds.b    1
05    bm_Depth          ds.b    1
06    bm_Pad            ds.w    1
08    bm_Planes         ds.b    32
28    bm_SIZEOF         ds.w    0
```

```
00    ra_MinX          ds.w    1
02    ra_MinY          ds.w    1
04    ra_MaxX          ds.w    1
06    ra_MaxY          ds.w    1
08    ra_SIZEOF        ds.w    0


;layers
00    lie_env            ds.b    52
34    lie_mem            ds.b    LH_SIZE
42    lie_FreeClipRects  ds.l    1
46    lie_blitbuff       ds.l    1
4A    lie_SIZEOF         ds.w    0

LMN_REGION    equ  -1


;memory
memnode_succ    ds.l    1
memnode_pred    ds.l    1
memnode_where   ds.l    1
memnode_how_big ds.l    1
memnode_SIZEOF  ds.w    0

;LayerInfo-Struktur
00    li_top_layer       ds.l    1        ;^Layer oben
04    li_check_lp        ds.l    1        ;System-Use:
08    li_obs             ds.l    1
0C    li_RP_ReplyPort    ds.b    MP_SIZE
2E    li_LockPort        ds.b    MP_SIZE
50    li_Lock            ds.b    1
51    li_broadcast       ds.b    1
52    li_locknest        ds.b    1
53    li_pad             ds.b    1
54    li_Locker          ds.l    1
58    li_bytereserved    ds.b    2
5A    li_wordreserved    ds.b    4
5E    li_longreserved    ds.b    4
62    li_LayerInfo_extra ds.l    1
66    li_SIZEOF          ds.w    0

NEWLAYERINFO_CALLED     equ 1

;rastport

00    tr_RasPtr          ds.l    1
04    tr_Size            ds.l    1
```

```
08   tr_SIZEOF          ds.w    0

00   gi_sprRsrvd        ds.b    1
01   gi_Flags           ds.b    1
02   gi_gelHead         ds.l    1
06   gi_gelTail         ds.l    1
0A   gi_nextLine        ds.l    1
0E   gi_lastColor       ds.l    1
12   gi_collHandler     ds.l    1
16   gi_leftmost        ds.w    1
18   gi_rightmost       ds.w    1
1A   gi_topmost         ds.w    1
1C   gi_bottommost      ds.w    1
1E   gi_firstBlissObj   ds.l    1
22   gi_lastBlissObj    ds.l    1
26   gi_SIZEOF          ds.w    0


RPB_FRST_DOT            equ     0
RPF_FRST_DOT            equ     1
RPB_ONE_DOT             equ     1
RPF_ONE_DOT             equ     2
RPB_DBUFFER             equ     2
RPF_DBUFFER             equ     4
RPB_AREAOUTLINE         equ     3
RPF_AREAOUTLINE         equ     8
RPB_NOCROSSFILL         equ     5
RPF_NOCROSSFILL         equ     32

RP_JAM1                 equ 0
RP_JAM2                 equ 1
RP_COMPLEMENT           equ 2
RP_INVERSVID            equ 4

RPB_TXSCALE             equ     0
RPF_TXSCALE             equ     1

;RastPort-Struktur
00   rp_Layer           ds.l    1       ;^Layer
04   rp_BitMap          ds.l    1       :^Bitmap
08   rp_AreaPtrn        ds.l    1       ;^Füllmuster
0C   rp_TmpRas          ds.l    1       ;^Zwischenpuffer
10   rp_AreaInfo        ds.l    1       ;^Info-Struktur
14   rp_GelsInfo        ds.l    1       ;^GelInfo-Struktur
18   rp_Mask            ds.b    1       ;Schreibmaske
19   rp_FgPen           ds.b    1       ;Vordergrund-Pen
1A   rp_BgPen           ds.b    1       ;Hintergrund-Pen
```

```
1B   rp_AOLPen        ds.b   1     ;Flood-Pen
1C   rp_DrawMode      ds.b   1     ;Zeichenmodus
1D   rp_AreaPtSz      ds.b   1     ;Worte Flood-Muster
1E   rp_Dummy         ds.b   1     ;Dummy
1F   rp_linpatcnt     ds.b   1     ;Poly-Count
20   rp_Flags         ds.w   1     ;System-Use
22   rp_LinePtrn      ds.w   1     ;Linienmuster
24   rp_cp_x          ds.w   1     ;Pen-Position
26   rp_cp_y          ds.w   1
28   rp_minterms      ds.b   8     ;Blitter-Control
30   rp_PenWidth      ds.w   1     ;Größe Pen
32   rp_PenHeight     ds.w   1
34   rp_Font          ds.l   1     ;^Font
38   rp_AlgoStyle     ds.b   1     ;Text-Parms:
39   rp_TxFlags       ds.b   1
3A   rp_TxHeight      ds.w   1
3C   rp_TxWidth       ds.w   1
3E   rp_TxBaseline    ds.w   1
40   rp_TxSpacing     ds.w   1
42   rp_RP_User       ds.l   1     ;^Reply-Port
46   rp_wordreserved  ds.b   14
54   rp_longreserved  ds.b   8
5C   rp_reserved      ds.b   8
64   rp_SIZEOF        ds.w   0

00   ai_VctrTbl       ds.l   1
04   ai_VctrPtr       ds.l   1
08   ai_FlagTbl       ds.l   1
0C   ai_FlagPtr       ds.l   1
10   ai_Count         ds.w   1
12   ai_MaxCount      ds.w   1
14   ai_FirstX        ds.w   1
16   ai_FirstY        ds.w   1
18   ai_SIZEOF        ds.w   0

ONE_DOTn        equ     1
ONE_DOT         equ     $2
FRST_DOTn       equ     0
FRST_DOT        equ     1


;REGIONS

00   rg_bounds            ds.b    ra_SIZEOF
08   rg_RegionRectangle   ds.l    1
0C   rg_SIZEOF            ds.w    0
```

```
00   rr_Next         ds.l    1
04   rr_Prev         ds.l    1
08   rr_bounds       ds.b    ra_SIZEOF
10   rr_SIZEOF       ds.w    0

;Sprites

00   ss_posctldata   ds.l    1      ;^Daten Sprite
04   ss_height       ds.w    1      ;Höhe
06   ss_x            ds.w    1      ;aktuelle Position X
08   ss_y            ds.w    1      ;                  Y
0A   ss_num          ds.w    1      ;Sprite-Nummer (0..7)
0C   ss_SIZEOF       ds.w    0


;Text

FS_NORMAL        equ     0
FSB_EXTENDED     equ     3
FSF_EXTENDED     equ     8
FSB_ITALIC       equ     2
FSF_ITALIC       equ     4
FSB_BOLD         equ     1
FSF_BOLD         equ     2
FSB_UNDERLINED   equ     0
FSF_UNDERLINED   equ     1

FPB_ROMFONT      equ     0
FPF_ROMFONT      equ     1
FPB_DISKFONT     equ     1
FPF_DISKFONT     equ     2
FPB_REVPATH      equ     2
FPF_REVPATH      equ     4
FPB_TALLDOT      equ     3
FPF_TALLDOT      equ     8
FPB_WIDEDOT      equ     4
FPF_WIDEDOT      equ     16
FPB_PROPORTIONAL equ     5
FPF_PROPORTIONAL equ     32
FPB_DESIGNED     equ     6
FPF_DESIGNED     equ     64
FPB_REMOVED      equ     7
FPF_REMOVED      equ     128

00   ta_Name         ds.l    1
04   ta_YSize        ds.w    1
06   ta_Style        ds.b    1
```

```
07   ta_Flags       ds.b   1
08   ta_SIZEOF      ds.w   0


14   tf_YSize       ds.w   1
16   tf_Style       ds.b   1
17   tf_Flags       ds.b   1
18   tf_XSize       ds.w   1
1A   tf_Baseline    ds.w   1
1C   tf_BoldSmear   ds.w   1
1E   tf_Accessors   ds.w   1
20   tf_LoChar      ds.b   1
21   tf_HiChar      ds.b   1
22   tf_CharData    ds.l   1
26   tf_Modulo      ds.w   1
28   tf_CharLoc     ds.l   1
2C   tf_CharSpace   ds.l   1
30   tf_CharKern    ds.l   1
34   tf_SIZEOF      ds.w   0


;View

V_PFBA          equ     $40
V_DUALPF        equ     $400
V_HIRES         equ     $8000
V_LAC           equ     4
V_HAM           equ     $800
V_SPRITES       equ     $4000

GENLOCK_VIDEO   equ 2

cm_Flags            ds.b   1
cm_Type             ds.b   1
cm_Count            ds.w   1
cm_ColorTable       ds.l   1
cm_SIZEOF           ds.w   0

;ViewPort-Struktur
00   vp_Next        ds.l   1     ;^Nachfolger
04   vp_ColorMap    ds.l   1     ;^
08   vp_DspIns      ds.l   1     ;
0C   vp_SprIns      ds.l   1
10   vp_ClrIns      ds.l   1
14   vp_UCopIns     ds.l   1
18   vp_DWidth      ds.w   1     ;Breite
1A   vp_DHeight     ds.w   1     ;Höhe
```

```
1C    vp_DxOffset     ds.w    1
1E    vp_DyOffset     ds.w    1
20    vp_Modes        ds.w    1
22    vp_reserved     ds.w    1
24    vp_RasInfo      ds.l    1
28    vp_SIZEOF       ds.w    0

00    v_ViewPort      ds.l    1
04    v_LOFCprList    ds.l    1
08    v_SHFCprList    ds.l    1
0C    v_DyOffset      ds.w    1
0E    v_DxOffset      ds.w    1
10    v_Modes         ds.w    1
12    v_SIZEOF        ds.w    0

00    cp_collPtrs     ds.l    1
04    cp_SIZEOF       ds.w    0


00    ri_Next         ds.l    1
04    ri_BitMap       ds.l    1
08    ri_RxOffset     ds.w    1
0A    ri_RyOffset     ds.w    1
0C    ri_SIZEOF       ds.w    0
```

5. Devices

```
;importiert von exec:

CMD_NONSTD      equ     9
IO_SIZE         equ     $20
IOSTD_SIZE      equ     $30
LN_SIZE         equ     $0E
MN_SIZE         equ     $14
TV_SIZE         equ     8
LIB_SIZE        equ     $22
MP_SIZE         equ     $22
pf_SIZEOF       equ     $E8
TC_SIZE         equ     $5C
LN_PRI          equ     9

;Audio

ADHARD_CHANNELS equ   4

ADALLOC_MINPREC equ   -128
```

```
ADALLOC_MAXPREC equu   127

CMD_NONSTD      equ    9

ADCMD_FREE      equ    9 ADCMD_SETPREC    equ    10
ADCMD_FINISH    equ    11
ADCMD_PERVOL    equ    12
ADCMD_LOCK      equ    13
ADCMD_WAITCYCLE equ    14

ADCMDB_NOUNIT   equ    5
ADCMDF_NOUNIT   equ    32
ADCMD_ALLOCATE  equ    ADCMDF_NOUNIT

ADIOB_PERVOL            equ    4
ADIOF_PERVOL            equ    16
ADIOB_SYNCCYCLE         equ    5
ADIOF_SYNCCYCLE         equ    32
ADIOB_NOWAIT            equ    6
ADIOF_NOWAIT            equ    64
ADIOB_WRITEMESSAGE      equ    7
ADIOF_WRITEMESSAGE      equ    128

ADIOERR_NOALLOCATION    equ    -10
ADIOERR_ALLOCFAILED     equ    -11
ADIOERR_CHANNELSTOLEN   equ    -12

20             ioa_AllocKey     ds.w     1
22             ioa_Data         ds.l     1
26             ioa_Length       ds.l     1
2A             ioa_Period       ds.w     1
2C             ioa_Volume       ds.w     1
2E             ioa_Cycles       ds.w     1
30             ioa_WriteMsg     ds.b     MN_SIZE
               ioa_SIZEOF       equ      $44

;bootblock

00             BB_ID            ds.b     4
04             BB_CHKSUM        ds.l     1
08             BB_DOSBLOCK      ds.l     1
               BB_ENTRY         equ      $0C
               BB_SIZE          equ      $0C

BOOTSECTS       equ    2

BBNAME_DOS      equ    444F5300  ; 'DOS'<<8
```

```
BBNAME_KICK          equ    4B49434B  ; 'KICK'

;CLIPBOARD

CBERR_OBSOLETEID     equ    1

00                 cu_Node            ds.b    LN_SIZE
0E                 cu_UnitNum         ds.l    1


00                 io_Message         ds.b    MN_SIZE
14                 io_Device          ds.l    1
18                 io_Unit            ds.l    1
1C                 io_Command         ds.w    1
1E                 io_Flags           ds.b    1
1F                 io_Error           ds.b    1
20                 io_Actual          ds.l    1
24                 io_Length          ds.l    1
28                 io_Data            ds.l    1
2C                 io_Offset          ds.l    1
30                 io_ClipID          ds.l    1
                   iocr_SIZEOF        equ     $34

PRIMARY_CLIP     equ    0

00                 sm_Msg             ds.b    MN_SIZE
14                 sm_Unit            ds.w    1
16                 sm_ClipID          ds.l    1
                   satisfyMsg_SIZEOF  equ     $1A


;CONSOLE

CD_ASKKEYMAP      equ    9
CD_SETKEYMAP      equ    10

SGR_PRIMARY       equ    0
SGR_BOLD          equ    1
SGR_ITALIC        equ    3
SGR_UNDERSCORE    equ    4
SGR_NEGATIVE      equ    7
SGR_BLACK         equ    30
SGR_RED           equ    31
SGR_GREEN         equ    32
SGR_YELLOW        equ    33
SGR_BLUE          equ    34
SGR_MAGENTA       equ    35
```

```
SGR_CYAN          equ   36
SGR_WHITE         equ   37
SGR_DEFAULT       equ   39
SGR_BLACKBG       equ   40
SGR_REDBG         equ   41
SGR_GREENBG       equ   42
SGR_YELLOWBG      equ   43
SGR_BLUEBG        equ   44
SGR_MAGENTABG     equ   45
SGR_CYANBG        equ   46
SGR_WHITEBG       equ   47
SGR_DEFAULTBG     equ   49
SGR_CLR0          equ   30
SGR_CLR1          equ   31
SGR_CLR2          equ   32
SGR_CLR3          equ   33
SGR_CLR4          equ   34
SGR_CLR5          equ   35
SGR_CLR6          equ   36
SGR_CLR7          equ   37
SGR_CLR0BG        equ   40
SGR_CLR1BG        equ   41
SGR_CLR2BG        equ   42
SGR_CLR3BG        equ   43
SGR_CLR4BG        equ   44
SGR_CLR5BG        equ   45
SGR_CLR6BG        equ   46
SGR_CLR7BG        equ   47

DSR_CPR           equ   6

CTC_HSETTAB       equ   0
CTC_HCLRTAB       equ   2
CTC_HCLRTABSALL   equ   5

TBC_HCLRTAB       equ   0
TBC_HCLRTABSALL   equ   3

;gameport

GPD_READEVENT     equ  9
GPD_ASKCTYPE      equ  10
GPD_SETCTYPE      equ  11
GPD_ASKTRIGGER    equ  12
GPD_SETTRIGGER    equ  13
GPTB_DOWNKEYS     equ  0
GPTF_DOWNKEYS     equ  1
```

```
GPTB_UPKEYS       equ  1
GPTF_UPKEYS       equ  2

00              gpt_Keys        ds.w    1
02              gpt_Timeout     ds.w    1
04              gpt_XDelta      ds.w    1
06              gpt_YDelta      ds.w    1
                gpt_SIZEOF      equ     8

GPCT_ALLOCATED          equ   -1
GPCT_NOCONTROLLER       equ   0
GPCT_MOUSE              equ   1
GPCT_RELJOYSTICK        equ   2
GPCT_ABSJOYSTICK        equ   3
GPDERR_SETCTYPE         equ   1

;input

IND_ADDHANDLER          equ    9
IND_REMHANDLER          equ   10
IND_WRITEEVENT          equ   11
IND_SETTHRESH           equ   12
IND_SETPERIOD           equ   13
IND_SETMPORT            equ   14
IND_SETMTYPE            equ   15
IND_SETMTRIG            equ   16


;INPUTEVENT

IECLASS_NULL            equ   0
IECLASS_RAWKEY          equ   1
IECLASS_RAWMOUSE        equ   2
IECLASS_EVENT           equ   3
IECLASS_POINTERPOS      equ   4
IECLASS_TIMER           equ   6
IECLASS_GADGETDOWN      equ   7
IECLASS_GADGETUP        equ   8
IECLASS_RequESTER       equ   9
IECLASS_MENULIST        equ   10
IECLASS_CLOSEWINDOW     equ   11
IECLASS_SIZEWINDOW      equ   12
IECLASS_REFRESHWINDOW   equ   13
IECLASS_NEWPREFS        equ   14
IECLASS_DISKREMOVED     equ   15
IECLASS_DISKINSERTED    equ   16
IECLASS_ACTIVEWINDOW    equ   17
```

```
IECLASS_INACTIVEWINDOW  equ   18

IECLASS_MAX             equ   $12

IECODE_UP_PREFIX        equ   $80
IECODEB_UP_PREFIX       equ   7
IECODE_KEY_CD_FIRST     equ   0
IECODE_KEY_CD_LAST      equ   $77
IECODE_COMM_CD_FIRST    equ   $78
IECODE_COMM_CODE_LAST   equ   $7F


IECODE_C0_FIRST         equ   $00
IECODE_C0_LAST          equ   $1F
IECODE_ASCII_FIRST      equ   $20
IECODE_ASCII_LAST       equ   $7E
IECODE_ASCII_DEL        equ   $7F
IECODE_C1_FIRST         equ   $80
IECODE_C1_LAST          equ   $9F
IECODE_LATIN1_FIRST     equ   $A0
IECODE_LATIN1_LAST      equ   $FF
IECODE_LBUTTON          equ   $68
IECODE_RBUTTON          equ   $69
IECODE_MBUTTON          equ   $6A
IECODE_NOBUTTON         equ   $FF

IECODE_NEWACTIVE        equ   1
IECODE_REQSET           equ   1
IECODE_REQCLEAR         equ   0


IequALIFIER_LSHIFT      equ   1
IequALIFIERB_LSHIFT     equ   0
IequALIFIER_RSHIFT      equ   2
IequALIFIERB_RSHIFT     equ   1
IequALIFIER_CAPSLOCK    equ   4
IequALIFIERB_CAPSLOCK   equ   2
IequALIFIER_CONTROL     equ   8
IequALIFIERB_CONTROL    equ   3
IequALIFIER_LALT        equ   16
IequALIFIERB_LALT       equ   4
IequALIFIER_RALT        equ   32
IequALIFIERB_RALT       equ   5
IequALIFIER_LCOMMAND    equ   64
IequALIFIERB_LCOMMAND   equ   6
IequALIFIER_RCOMMAND    equ   128
IequALIFIERB_RCOMMAND   equ   7
```

```
IequALIFIER_NUMERICPAD          equ     $0100
IequALIFIERB_NUMERICPAD         equ     8
IequALIFIER_REPEAT              equ     $0200
IequALIFIERB_REPEAT             equ     9
IequALIFIER_INTERRUPT           equ     $0400
IequALIFIERB_INTERRUPT          equ     10
IequALIFIER_MULTIBROADCAST      equ     $0800
IequALIFIERB_MULTIBROADCAST     equ     11
IequALIFIER_LBUTTON             equ     $1000
IequALIFIERB_LBUTTON            equ     12
IequALIFIER_RBUTTON             equ     $2000
IequALIFIERB_RBUTTON            equ     13
IequALIFIER_MBUTTON             equ     $4000
IequALIFIERB_MBUTTON            equ     14
IequALIFIER_RELATIVEMOUSE       equ     $8000
IequALIFIERB_RELATIVEMOUSE      equ     15


00              ie_NextEvent    ds.l    1
04              ie_Class        ds.b    1
05              ie_SubClass     ds.b    1
06              ie_Code         ds.w    1
08              ie_Qualifier    ds.w    1
                ie_EventAddress equ     $0A
0A              ie_X            ds.w    1
0c              ie_Y            ds.w    1


0E              ie_TimeStamp    ds.b    TV_SIZE
16              ie_SIZEOF       ds.w    0

;KEYBOARD

KBD_READEVENT           equ     9
KBD_READMATRIX          equ     10
KBD_ADDRESETHANDLER     equ     11
KBD_REMRESETHANDLER     equ     12
KBD_RESETHANDLERDONE    equ     13

;Keymap

00              km_LoKeyMapTypes ds.l    1
04              km_LoKeyMap      ds.l    1
08              km_LoCapsable    ds.l    1
0C              km_LoRepeatable  ds.l    1
10              km_HiKeyMapTypes ds.l    1
14              km_HiKeyMap      ds.l    1
```

```
18              km_HiCapsable    ds.l     1
1C              km_HiRepeatable  ds.l     1
                km_SIZEOF        equ      $20


KCB_NOP          equu   7
KCF_NOP          equ    $80

KC_NOQUAL        equ    0
KC_VANILLA       equ    7
KCF_SHIFT        equ    1
KCF_ALT          equ    2
KCB_CONTROL      equ    2
KCF_CONTROL      equ    4
KCB_DOWNUP       equ    3
KCF_DOWNUP       equ    8

KCB_STRING       equ    6
KCB_STRING       equ    64


;Narrator

DEFPITCH         equ        110
DEFRATE          equ        150
DEFVOL           equ        64
DEFFREQ          equ        22200
NATURALF0        equ        0
ROBOTICF0        equ        1
MALE             equ        0
FEMALE           equ        1
DEFSEX           equ        MALE
DEFMODE          equ        NATURALF0

MINRATE          equ        40
MAXRATE          equ        400
MINPITCH         equ        65
MAXPITCH         equ        320
MINFREQ          equ        5000
MAXFREQ          equ        28000
MINVOL           equ        0
MAXVOL           equ        64

ND_NotUsed       equ        -1
ND_NoMem         equ        -2
ND_NoAudLib      equ        -3
ND_MakeBad       equ        -4
```

```
ND_UnitErr      equ      -5
ND_CantAlloc    equ      -6
ND_Unimpl       equ      -7
ND_NoWrite      equ      -8
ND_Expunged     equ      -9
ND_PhonErr      equ      -20
ND_RateErr      equ      -21
ND_PitchErr     equ     -22
ND_SexErr       equ     -23
ND_ModeErr      equ     -24
ND_FreqErr      equ     -25
ND_VolErr       equ     -26


30             NDI_RATE        ds.w     1
32             NDI_PITCH       ds.w     1
34             NDI_MODE        ds.w     1
36             NDI_SEX         ds.w     1
38             NDI_CHMASKS     ds.l     1
3C             NDI_NUMMASKS    ds.w     1
3E             NDI_VOLUME      ds.w     1
40             NDI_SAMPFREQ    ds.w     1
42             NDI_MOUTHS      ds.b     1
43             NDI_CHANMASK    ds.b     1
44             NDI_NUMCHAN     ds.b     1
45             NDI_PAD         ds.b     1
               NDI_SIZE        equ      $46


46             MRB_WIDTH       ds.b     1
47             MRB_HEIGHT      ds.b     1
48             MRB_SHAPE       ds.b     1
49             MRB_PAD         ds.b     1
4A             MRB_SIZE        equ      $4A


;PARALLEL

ParErr_DevBusy          equ    1
ParErr_BufTooBig        equ    2
ParErr_InvParam         equ    3
ParErr_LineErr          equ    4
ParErr_NotOpen          equ    5
ParErr_PortReset        equ    6
ParErr_InitErr          equ    7
```

```
PDCMD_QUERY             equ   CMD_NONSTD
PDCMD_SETPARAMS         equ   CMD_NONSTD+1
Par_DEVFINISH           equ   10

PARB_SHARED     equ     5
PARF_SHARED     equ     32
PARB_RAD_BOOGIE equ     3
PARF_RAD_BOOGIE equ     8
PARB_EOFMODE    equ     1
PARF_EOFMODE    equ     2

IOPARB_QUEUED   equ     6
IOPARF_QUEUED   equ     128
IOPARB_ABORT    equ     5
IOPARF_ABORT    equ     32
IOPARB_ACTIVE   equ     4
IOPARF_ACTIVE   equ     16
IOPTB_RWDIR     equ     3 IOPTF_RWDIR      equ      8
IOPTB_PBUSY     equ     2
IOPTF_PBUSY     equ     4
IOPTB_PAPEROUT  equ     1
IOPTF_PAPEROUT  equ     2
IOPTB_PSEL      equ     0
IOPTF_PSEL      equ     1

00             PTERMARRAY_0    ds.l    1
04             PTERMARRAY_1    ds.l    1
08             PTERMARRAY_SIZE ds.w    0

30             IO_PEXTFLAGS    ds.l    1
34             IO_PARSTATUS    ds.b    1
35             IO_PARFLAGS     ds.b    1
36             IO_PTERMARRAY   ds.b    PTERMARRAY_SIZE
3E             IOEXTPar_SIZE   equ     $3E


;Serial

SER_CTL         equ     $11130000
SER_DBAUD       equ     9600


SDCMD_QUERY     equ     9
SDCMD_BREAK     equ     10
SDCMD_SETPARAMS equ     CMD_NONSTD+2
SER_DEVFINISH   equ     11
```

```
SERB_XDISABLED  equ     7
SERF_XDISABLED  equ     128
SERB_EOFMODE    equ     6
SERF_EOFMODE    equ     64
SERB_SHARED     equ     5
SERF_SHARED     equ     32
SERB_RAD_BOOGIE equ     4
SERF_RAD_BOOGIE equ     16
SERB_QUEUEDBRK  equ     3
SERF_QUEUEDBRK  equ     8
SERB_7WIRE      equ     2
SERF_7WIRE      equ     4
SERB_PARTY_ODD  equ     1
SERF_PARTY_ODD  equ     2
SERB_PARTY_ON   equ     0
SERF_PARTY_ON   equ     1

IOSERB_QUEUED    equ     6
IOSERF_QUEUED    equ     64
IOSERB_ABORT     equ     5
IOSERF_ABORT     equ     32
IOSERB_ACTIVE    equ     4
IOSERF_ACTIVE    equ     16
IOSTB_XOFFREAD   equ     4
IOSTF_XOFFREAD   equ     16
IOSTB_XOFFWRITE  equ     3
IOSTF_XOFFWRITE  equ     8
IOSTB_READBREAK  equ     2
IOSTF_READBREAK  equ     4
IOSTB_WROTEBREAK equ     1
IOSTF_WROTEBREAK equ     2
IOSTB_OVERRUN    equ     0
IOSTF_OVERRUN    equ     1


00             TERMARRAY_0      ds.l     1
04             TERMARRAY_1      ds.l     1
               TERMARRAY_SIZE   equ      8


30             IO_CTLCHAR       ds.l     1
34             IO_RBUFLEN       ds.l     1
38             IO_EXTFLAGS      ds.l     1
3C             IO_BAUD          ds.l     1
40             IO_BRKTIME       ds.l     1
44             IO_TERMARRAY     ds.b     TERMARRAY_SIZE
4C             IO_READLEN       ds.b     1
```

```
4D          IO_WRITELEN     ds.b    1
4E          IO_STOPBITS     ds.b    1
4F          IO_SERFLAGS     ds.b    1
50          IO_STATUS       ds.w    1
            IOEXTSER_SIZE   equ     $52


SerErr_DevBusy          equ      1
SerErr_BaudMismatch     equ      2
SerErr_InvBaud          equ      3
SerErr_BufErr           equ      4
SerErr_InvParam         equ      5
SerErr_LineErr          equ      6
SerErr_NotOpen          equ      7
SerErr_PortReset        equ      8
SerErr_ParityErr        equ      9
SerErr_InitErr          equ     10
SerErr_TimerErr         equ     11
SerErr_BufOverflow      equ     12
SerErr_NoDSR            equ     13
SerErr_NoCTS            equ     14
SerErr_DetectedBreak    equ     15


;timer

UNIT_MICROHZ    equ    0
UNIT_VBLANK     equ    1

00          TV_SECS         ds.l    1
04          TV_MICRO        ds.l    1
            TV_SIZE         equ     8

20          IOTV_TIME       ds.b    TV_SIZE
28          IOTV_SIZE       equ     $20

TR_ADDRequEST    equ     9
TR_GETSYSTIME    equ     10
TR_SETSYSTIME    equ     11


PRD_RAWWRITE     equ     9
PRD_PRTCOMMAND   equ     10
PRD_DUMPRPORT    equ     11

aRIS    equ    0
aRIN    equ    1
```

```
aIND     equ    2
aNEL     equ    3
aRI      equ    4

aSGR0    equ    5
aSGR3    equ    6
aSGR23   equ    7
aSGR4    equ    8
aSGR24   equ    9
aSGR1    equ   10
aSGR22   equ   11
aSFC     equ   12
aSBC     equ   13

aSHORP0  equ   14
aSHORP2  equ   15
aSHORP1  equ   16
aSHORP4  equ   17
aSHORP3  equ   18
aSHORP6  equ   19
aSHORP5  equ   20

aDEN6    equ   21
aDEN5    equ   22
aDEN4    equ   23
aDEN3    equ   24
aDEN2    equ   25
aDEN1    equ   26

aSUS2    equ   27
aSUS1    equ   28
aSUS4    equ   29
aSUS3    equ   30
aSUS0    equ   31
aPLU     equ   32
aPLD     equ   33

aFNT0    equ   34
aFNT1    equ   35
aFNT2    equ   36
aFNT3    equ   37
aFNT4    equ   38
aFNT5    equ   39
aFNT6    equ   40
aFNT7    equ   41
aFNT8    equ   42
aFNT9    equ   43
```

```
aFNT10  equ   44

aPROP2  equ   45
aPROP1  equ   46
aPROP0  equ   47
aTSS    equ   48

aJFY5   equ   49
aJFY7   equ   50
aJFY6   equ   51
aJFY0   equ   52
aJFY2   equ   53
aJFY3   equ   54

aVERP0  equ   55
aVERP1  equ   56
aSLPP   equ   57
aPERF   equ   58
aPERF0  equ   59

aLMS    equ   60
aRMS    equ   61
aTMS    equ   62
aBMS    equ   63
aSTBM   equ   64
aSLRM   equ   65
aCAM    equ   66

aHTS    equ   67
aVTS    equ   68
aTBC0   equ   69
aTBC3   equ   70
aTBC1   equ   71
aTBC4   equ   72
aTBCALL equ   73
aTBSALL equ   74
aEXTEND equ   75


20             io_PrtCommand   ds.w    1
22             io_Parm0        ds.b    1
23             io_Parm1        ds.b    1
24             io_Parm2        ds.b    1
25             io_Parm3        ds.b    1
               iopcr_SIZEOF    equ     $26

20             io_RastPort     ds.l    1
```

```
24              io_ColorMap     ds.l    1
28              io_Modes        ds.l    1
2C              io_SrcX         ds.w    1
2E              io_SrcY         ds.w    1
30              io_SrcWidth     ds.w    1
32              io_SrcHeight    ds.w    1
34              io_DestCols     ds.l    1
38              io_DestRows     ds.l    1
3C              io_Special      ds.w    1
                iodrpr_SIZEOF   equ     $3E

SPECIAL_MILCOLS          equ   1
SPECIAL_MILROWS          equ   2
SPECIAL_FULLCOLS         equ   4
SPECIAL_FULLROWS         equ   8
SPECIAL_FRACCOLS         equ   16
SPECIAL_FRACROWS         equ   32

22              dd_Segment      ds.l    1
26              dd_ExecBase     ds.l    1
2A              dd_CmdVectors   ds.l    1
2E              dd_CmdBytes     ds.l    1
32              dd_NumCommands  ds.w    1
                dd_SIZEOF       equ     $34

du_Flags         equ   LN_PRI

IOB_QUEUED       equ     4
IOF_QUEUED       equ     16
IOB_CURRENT      equ     5
IOF_CURRENT      equ     32
IOB_SERVICING    equ     6
IOF_SERVICING    equ     64
IOB_DONE         equ     7
IOF_DONE         equ     128

DUB_STOPPED      equ     0
DUF_STOPPED      equ     10

P_PRIORITY       equ     0
P_STKSIZE        equ     $800

PB_IOR0          equ     0
PF_IOR0          equ     1
PB_IOR1          equ     1
PF_IOR1          equ     2
PB_EXPUNGED      equ     7
```

```
PF_EXPUNGED      equ     128


34            pd_Unit           ds.b    MP_SIZE
56            pd_PrinterSegment ds.l    1
5A            pd_PrinterType    ds.w    1
5C            pd_SegmentData    ds.l    1
60            pd_PrintBuf       ds.l    1
64            pd_PWrite         ds.l    1
68            pd_PBothReady     ds.l    1

PPCB_GFX         equ     0
PPCF_GFX         equ     1
PPCB_COLOR       equ     1
PPCF_COLOR       equ     2

PPC_BWALPHA      equ     0
PPC_BWGFX        equ     1
PPC_COLORGFX     equ     3

PCC_BW           equ     1
PCC_YMC          equ     2
PCC_YMC_BW       equ     3
PCC_YMCB         equ     4

00            ped_PrinterName  ds.l   1
04            ped_Init         ds.l   1
08            ped_Expunge      ds.l   1
0C            ped_Open         ds.l   1
10            ped_Close        ds.l   1
14            ped_PrinterClass ds.b   1
15            ped_ColorClass   ds.b   1
16            ped_MaxColumns   ds.b   1
17            ped_NumCharSets  ds.b   1
18            ped_NumRows      ds.w   1
1A            ped_MaxXDots     ds.l   1
1E            ped_MaxYDots     ds.l   1
22            ped_XDotsInch    ds.w   1
24            ped_YDotsInch    ds.w   1
26            ped_Commands     ds.l   1
2A            ped_DoSpecial    ds.l   1
2E            ped_Render       ds.l   1
32            ped_TimeoutSecs  ds.l   1
              ped_SIZEOF       equ    $36

00            ps_NextSegment   ds.l   1
04            ps_runAlert      ds.l   1
```

```
08              ps_Version         ds.w     1
0A              ps_Revision        ds.w     1
                ps_PED             equ      $0C


SPECIAL_ASPECT            equ    $80
SPECIAL_DENSITYMASK       equ    $F00
SPECIAL_DENSITY1          equ    $100
SPECIAL_DENSITY2          equ    $200
SPECIAL_DENSITY3          equ    $300
SPECIAL_DENSITY4          equ    $400


PDERR_CANCEL              equ    1
PDERR_NOTGRAPHICS         equ    2
PDERR_INVERTHAM           equ    3
PDERR_BADDIMENSION        equ    4
PDERR_DIMENSIONOVFLOW     equ    5
PDERR_INTERNALMEMORY      equ    6
PDERR_BUFFERMEMORY        equ    7


;TRACKDISK

NUMCYLS         equ    80
MAXCYLS         equ    NUMCYLS+20
NUMSECS         equ    11
NUMHEADS        equ    2
MAXRETRY        equ    10
NUMTRACKS       equ    NUMCYLS*NUMHEADS
NUMUNITS        equ    4


TD_SECTOR       equ    512
TD_SECSHIFT     equ    9

TDB_EXTCOM      equ    15
TDF_EXTCOM      equ    $8000

;Commands
TD_MOTOR        equ    9
TD_SEEK         equ    10
TD_FORMAT       equ    11
TD_REMOVE       equ    12
TD_CHANGENUM    equ    13
TD_CHANGESTATE  equ    14
TD_PROTSTATUS   equ    15
```

```
TD_LASTCOMM      equ      15

;Extended Commands (mit extendedt IORequest-Block)
ETD_WRITE        equ    3
ETD_READ         equ    2
ETD_MOTOR        equ    9
ETD_SEEK         equ    10
ETD_FORMAT       equ    11
ETD_UPDATE       equ    4
ETD_CLEAR        equ    5

;IORequest-Block-Extension
30             IOTD_COUNT      ds.l     1
34             IOTD_SECLABEL   ds.l     1
               IOTD_SIZE       equ      $38

TD_LABELSIZE            equ   16

;Error-Codes (in IOActual)
TDERR_NotSpecified      equ   20
TDERR_NoSecHdr          equ   21
TDERR_BadSecPreamble    equ   22
TDERR_BadSecID          equ   23
TDERR_BadHdrSum         equ   24
TDERR_BadSecSum         equ   25
TDERR_TooFewSecs        equ   26
TDERR_BadSecHdr         equ   27
TDERR_WriteProt         equ   28
TDERR_DiskChanged       equ   29
TDERR_SeekError         equ   30
TDERR_NoMem             equ   31
TDERR_BadUnitNum        equ   32
TDERR_BadDriveType      equ   33
TDERR_DriveInUse        equ   34
```

Fehlermeldungen

Anhang 5

Guru-Meditation Error Codes

Die sechzehn Ziffern eines Guru-Fehlercodes lassen sich nach folgendem Schema unterteilen:

AABBCCCC.EEEEEEEE

Die ersten beiden Ziffern (A) kennzeichnen das Subsystem, die nächsten beiden (B) den grundsätzlichen Fehler und die Vierergruppe (C) den spezifizierten Fehler. Jede Subsystem-Kennzeichnung (A) hat jeweils anders spezifizierte Vierergruppen (C). Die letzten acht Ziffern bestimmen schließlich die Adresse (Hexadezimal) des Tasks, der den Absturtz verursacht hat.

Mit der Anzeige des Fehlercodes fordert der Computer den Benutzer auf, die linke Maustaste zu drücken. Mit der rechten Maustaste wird in bestimmten Fällen das neuerliche Booten umgangen. Ist die erste Ziffer kleiner oder gleich 7, springt der Computer mit dem Druck auf die rechte Maustaste direkt ins Programm zurück. Ist die erste Ziffer größer 7, ist eine Fortführung des Programms nicht mehr möglich.

Prozessor-Fehler

Der Fehlercode 00 in der ersten Zahlengruppe (Kennung A) hat eine Ausnahmestellung inne. Er steht für einen CPU-Fehler. Diese Meldungen sind durch den 68000-Prozessor und nicht durch die Amiga-Systemsoftware definiert. Dem Prozessor stehen für die Fehlerbehandlung 256 Vektoren zur Verfügung. Die ersten 64 dieser Vektoren sind durch den Prozessor definiert. Die oberen 192 Vektoren können durch den Benutzer definiert werden.

00 00 0002	Bus-Error Timingfehler auf Adress- oder Datenbus
00 00 0003	Address Error Adressierungsfehler
00 00 0004	Illegal Instruction Unzulässige Instruktion

00 00 0005	Devide by Zero Division durch Null
00 00 0006	CHK Instruktion CHK testet Register gegen die Grenzen des Zahlenbereichs. Der Fehler tritt auf, wenn die zulässigen Grenzen nicht eingehalten werden.
00 00 0007	TRAPV Instruction TRAPV verzweigt auf Trap-Vektor, wenn V-Flag gesetzt.
00 00 0008	Privilege Violation Privilegverletzung
00 00 0009	Trace Einzelschritt-Modus
00 00 000A	OP Code 1010 Unbenutzter Op-Code
00 00 000B	OP Code 1111 Unbenutzter Op-Code

Amiga-Systemfehlermeldungen

Alle anderen Guru-Meditations bezeichnen mit den ersten beiden Stellen (A) den Betriebssystemteil, in dem der Fehler aufgetreten ist.

Die Libraries haben in der ersten Position die Null

01 BB CCCC	Exec
02 BB CCCC	Graphics
03 BB CCCC	Layers
04 BB CCCC	Intuition
05 BB CCCC	Math
06 BB CCCC	Clist
07 BB CCCC	DOS
08 BB CCCC	RAM
09 BB CCCC	Icon
0A BB CCCC	Expansion

Die Devices sind durch die Eins gekennzeichnet

10 BB CCCC	Audio
11 BB CCCC	Console
12 BB CCCC	GamePort
13 BB CCCC	Keyboard
14 BB CCCC	TrackDisk
15 BB CCCC	Timer

Die Zwei definiert Resource

20 BB CCCC	CIA
21 BB CCCC	Disk
22 BB CCCC	Misc

Die »Sonstigen« haben die Drei in der ersten Ziffer

30 BB CCCC	Bootstrap
31 BB CCCC	Workbench
32 BB CCCC	DiskCopy

Übergeordnete Fehler

AA 00 CCCC	Fehler nicht zuzuordnen
AA 01 CCCC	Insufficient Memory Speicherplatzmangel
AA 02 CCCC	MakeLibrary Error Library kann nicht erzeugt werden
AA 03 CCCC	OpenLibrary Error Library kann nicht geöffnet werden
AA 04 CCCC	OpenDevice Error Device kann nicht geöffnet werden
AA 05 CCCC	OpenResource Error Nichtreagieren eines Hardware-Bausteins
AA 06 CCCC	I/O Error Ein-/Ausgabefehler
AA 07 CCCC	No Signal Signal fehlt

1. Exec Library-Codes

81 00 0001	68000 Exception Vector Checksum Prüfsummenfehler bei Ausnahmebehandlung des Prozessors
81 00 0002	ExecBase Checksum Prüfsummenfehler der Staradresse des Exec
81 00 0003	Library Checksum Error Prüfsummenfehler bei der Library
81 00 0004	No Memory to Make Library Mangelnder Speicherplatz für Library
81 00 0005	Corrupted Memory List Zerstörte Speicherverwaltungsliste
81 00 0006	No Memory For Interrupt Servers Mangelnder Speicherplatz für Interruptbehandlung
81 00 0007	initAPTR Zeigerfehler
81 00 0008	Semaphore Corrupt Semaphore zerstört
81 00 0009	Free Twice Speicherplatz zweimal freigegeben
81 00 000A	Bogus Exception Es wurden reservierte Vektoren verwendet

2. Graphics Library-Codes

82 01 0001	No Memory for Copper Display List Mangelnder Speicherplatz für die Copperliste
82 01 0002	No Memory for Copper Instruction List Mangelnder Speicherplatz für die Copper Instruction-Liste
82 00 0003	Copper List Overload Copperliste ist voll

82 00 0004	Copper Intermediate List Overload Struktur der Copperliste zerstört
82 01 0005	No Memory for Copper List Head Mangelnder Speicherplatz für den Kopf der Copperliste
82 01 0006	Long Frame, No Memory Mangelnder Speicherplatz für die Copper-Liste I bei Interlace.
82 01 0007	Short Frame, No Memory Mangelnder Speicherplatz für die Copper-Liste I bei Interlace.
82 01 0008	No Memory For Flood Fill Mangelnder Speicherplatz zum Ausführen des Fill-Befehls
82 01 0009	Text, No Memory For TmpRas Mangelnder Speicherplatz zum Anlegen der temporären Datei TmpRas (Temporary raster work area)
82 01 000A	No Memory for BltBitMap Mangelnder Speicherplatz für die Blitter-Bitmap
82 01 000B	Region Memory Speicherbereich falsch angegeben
82 01 0030	MakeVPort Fehler beim Einrichten des ViewPort
82 01 1234	GfxNoLCM Zwischenspeicherbereich nicht frei

3. Layers Library-Codes

83 01 0001	LayersNoMem Mangelnder Speicherplatz für die Layers

4. Intuition Library-Codes

84 00 0001	Unknown Gadget Type Unbekannter Gadget Typ
04 00 0001	 Wie oben, doch abfangbar
84 01 0002	No Memory to create Port Mangelnder Speicherplatz für neuen Port
84 01 0003	Item Plane Alloc, No Memory Mangelnder Speicherplatz für Darstellung der Menüleiste
84 01 0004	Sub Alloc, No Memory Mangelnder Speicherplatz für Darstellung der Untermenüs
84 01 0005	Plane Alloc, No Memory Mangelnder Speicherplatz für die Kopfzeile des Menüs
84 00 0006	Item Box Top Less Than Real Zero Die obere Grenze einer Item-Box liegt unter der absoluten Null-Position
84 01 0007	No Memory To Open Screen Mangelnder Speicherplatz zum Öffnen eines Screens
84 01 0008	Open Screen, Raster Alloc, No Memory Mangelnder Speicherplatz für RastPort
84 00 0009	Open Sys Screen, Unknown Type Unbekannter Screen-Typ
84 01 000A	Add SW Gadgets, No memory Mangelnder Speicherplatz für Gadget
84 01 000B	No Memory to Open Window Mangelnder Speicherplatz zum Öffnen eines Windows
84 00 000C	Bad State Return Entering Intuition Fehlerhafte Statusangabe beim Öffnen von Intuition

84 00 000D	Bad Message Received by IDCMP Fehlermeldung der "Intuition Direct Communication Message Ports"
84 00 000E	Wierd Echo Causing Incomprehension Mangelnder Speicherplatz für Zugriff auf die "Distant Echo List"
84 00 000F	Could not Open The Console Device Fehler beim Öffnen des Console Device

5. DOS Library-Codes

07 01 0001	No Memory At Startup Mangelnder Speicherplatz bei Startup
07 00 0002	EndTask didn't EndTask hat fehlerhaft oder nicht gewirkt
07 00 0003	Qptk Failure Fehler beim Übertragen eines Datenpaketes
07 00 0004	Unexpected Packet Received Empfang eines Datenpaketes, das nicht erwartet wurde
07 00 0005	Freevec Failed Freevec hat fehlerhaft oder nicht gewirkt
07 00 0006	Disk Block Sequence Error Fehler bei einer Disk-Block-Sequenz
07 00 0007	Bitmap Corrupt Fehlerhafte oder zerstörte Bitmap
07 00 0008	Key already Free File-Nummer (Key-Nummer) bereits gelöscht
07 00 0009	Invalid Checksum Unzulässige Prüfsumme
07 00 000A	Disk Error Diskettenfehler

07 00 000B Key Out Of Range
File-Nummer (Key-Nummer) außerhalb des zulässigen Bereiches

07 00 000C Bad Overlay
Overlay Hunk nicht in Ordnung

6. RAM Library-Codes

08 00 0001 Bad Segment List
Fehlerhafte Speicherverwaltungsliste

7. Expansion Library-Codes

0A 00 0001 Bad Expansion Free
Hard- oder Softwarefehler bei einer Erweiterung

8. TrackDisk Device Codes

14 00 0001 Calibrate: Seek Error
Fehler beim Suchen auf Diskette

14 00 0002 Delay: Error On Timer Wait
Fehler beim Warten auf einen Timer-Impuls

9. Timer Device-Codes

15 00 0001 Bad Request
Fehler beim Zugriffsversuch

15 00 0002 Bad Supply
Fehlsteuerung durch Netzfrequenz (Frequenz instabil)

10. Disk Resource-Codes

21 00 0001 Get Unit: Already has Disk
Fehlerhaftes DiskChange-Signal

21 00 0002 Interrupt: No Active Unit
Kein aktives Laufwerk vorhanden

11. BootStrap-Codes

30 00 0001 Boot Code Returned an Error
DOS-Library nicht gefunden

Quellennachweis der Bilder

Anhang 6

Bild 1.1: Amiga Assembler-Buch, P. Wollschlaeger,
Markt&Technik Verlag

Bild 2.1: Amiga 3D-Grafik und Animation, A. Plenge
Markt&Technik Verlag

Bild 2.2: Amiga Programmierpraxis Intuition, P. Wollschlaeger
Markt&Technik Verlag

Bild 2.3: Amiga Programmierpraxis Intuition, P. Wollschlaeger
Markt&Technik Verlag

Bild 2.4: Amiga Programmierpraxis Intuition, P. Wollschlaeger
Markt&Technik Verlag

Bild 4.10: C64/C128 Alles über Maschinensprache,
F. Riemenschneider, Markt&Technik Verlag

Stichwortverzeichnis

C

D

E

F

P

Q

R

S

W

X

Y

Z

Frank Riemenschneider

AMIGA
Programmieren in Maschinensprache

Genauso wie sein Vorgänger C64 wird der Amiga erst durch die Programmierung in Maschinensprache richtig ausgenutzt. Während das Amiga-Basic neben der viel zu geringen Geschwindigkeit vor allen Dingen dadurch auffällt, daß man Features wie Gadgets oder Untermenüs gar nicht erst programmieren kann, stören bei der Sprache C der langatmige Kompilierungsprozeß sowie die wenig kompakten und damit langsamen Kompilate.

Da jetzt aber viele Amiga-Besitzer sagen: Assembler bzw. Maschinensprache ist mir zu schwierig, ging der Autor dieses Buches einen neuen Weg. Er führt den Leser über Theorie und anschließende, vertiefende Praxis zum Erfolg, wobei alle Beispielprogramme auf der beiliegenden Diskette enthalten sind.

Im ersten Kapitel finden Sie alle 68000-Befehle in Funktion, Wirkung und anhand von Beispielen erläutert. Alle weiteren Kapitel beschäftigen sich mit ausführlichen, praktischen Übungen. Durch die zahlreiche Verwendung von Library-Funktionen wird die Maschinensprache dabei so leicht anwendbar wie auf keinem anderen Computer!

Folgende Themen werden umfassend beschrieben:

- Benutzerfreundliche Programmierung durch Intuition
- Grafik- und Chartprogrammierung
- Rechnen mit Fließkommazahlen
- Ausgabeoperationen mit dem Amiga-DOS
- Programmierung der Devices

Zu den einzelnen Kapiteln existieren umfangreiche Beispielprogramme, wie sie bisher in Assemblerbüchern noch nicht veröffentlicht wurden. Besonderer Wert wurde darauf gelegt, daß sich die ausführlich kommentierten Routinen auch in eigenen Programmen sinnvoll einsetzen lassen. Neben einer mausgesteuerten Luxus-Wertetabelle und einem 3-D-Chart-Grafikprogramm, das sämtliche Diskettenzeichensätze der Workbench verwendet, wird auch eine Dateiauswahl vorgestellt, wie sie in professionellen Programmen (z.B. Deluxe-Paint) verwendet wird. Weiterhin finden Sie die schnellste universelle Ellipsenroutine für den Amiga und ein Programm zum Ausdruck einer Hardcopy, vom Briefmarkenformat bis zum Mehrfachen des Bildschirms.

Umsteiger vom C64 oder C128 finden zahlreiche Hinweise an den Stellen, an denen Parallelen zu diesen Rechnern zu entdecken sind oder an denen relevante Unterschiede auftreten (z.B. Aufbau der Fließkommazahlen).

Abgerundet wird das Buch durch einen umfangreichen Anhang, aus dem neben allen Library-Funktionen, Strukturen und Assembler-Befehlen auch der Aufbau der Startup-Sequenz für den Programmstart von der Workbench bzw. dem CLI sowie der Aufbau eines .info-Files hervorgeht.

Die Begleitdiskette:

Sie enthält alle Beispielprogramme als Quelltext und Objectcode, Makro-Definitionen für wichtige Grafikfunktionen.

Hardware-Anforderungen:

Amiga 500, 1000 oder 2000

Software-Anforderungen:

Assembler-System: z.B. Devpac von Hi-Soft, Markt&Technik Verlag, Best.-Nr. 54131

ISBN 3-89090-712-1

4 001057 907126

Markt&Technik

DM 69,-
sFr 63,50
öS 538,-